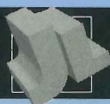
THE PETER NORTON PROGRAMMING LIBRARY

David I. Schneider e
Peter Norton Computing Group

- Più di 100 programmi pronti all'uso
- Consigli avanzati sulla grafica, il suono e la gestione dei file
- Applicazioni professionali, scientifiche e matematiche

PROGRAMMARE IN

David I. Schneider e
Peter Norton Computing Group

Titolo originale
QBASIC PROGRAMMING

Original English Language edition published by BRADY PUBLISHING
A Division of Simon & Schuster, Englewood Cliffs, New Jersey, USA

Una coedizione di
Prentice Hall International, Hemel Hempstead, England
Jackson Libri S.r.l. , Milano, Italia

Traduzione e impaginazione con tecniche di desktop publishing
LINK s.r.l.

Redattore di collana
Giovanni Perotti

Copertina
Laura Berettini

Prima edizione 1992

Ristampa

9 8 7 6 5

1996 1995 1994

INDICE

Capitolo 4 Gestione dei dati ... 33

Capitolo 5 Decisioni e ripetizioni ... 71

INTRODUZIONE

Il BASIC è uno dei linguaggi più diffusi nel mondo del personal computer. Gli utenti che scrivono dei programmi per diletto apprezzano il BASIC perché è facile da imparare e nello stesso tempo perché consente di sfruttare pienamente tutte le caratteristiche dell'elaboratore utilizzato, mentre i programmatori professionali possono sfruttare la velocità e la flessibilità proprie di questo linguaggio.

Negli ultimi 20 anni sono stati sviluppati diversi linguaggi che forniscono istruzioni per una programmazione più accurata e precisa. Linguaggi come il C e il Pascal dispongono di strutture di controllo che permettono di sviluppare programmi ben progettati e facili da revisionare. Sfortunatamente, queste strutture di controllo non sono disponibili nella maggior parte delle versioni del BASIC apparse sul mercato dopo la nascita di questo linguaggio, avvenuta nel 1964.

QBasic, la versione del BASIC incorporata nel sistema operativo MS-DOS 5.0, ha ereditato le caratteristiche di tutti questi linguaggi. È facile da imparare e utilizzare come le versioni standard del BASIC, e dispone di strutture di controlle proprie dei linguaggi 'seri' come il C e il Pascal. Seguono alcune delle caratteristiche del QBasic:

- funzionalità complete di editing;
- compilazione rapida ed efficiente;
- procedure;

- funzioni su più righe;
- possibilità di passare dei parametri;
- variabili locali, statiche, e globali
- ricorsività reale;
- blocchi IF...THEN...ELSEIF...ELSE;
- blocchi SELECT CASE;
- cicli DO;
- numeri di riga opzionali;
- variabili definite dall'utente.

UNA PANORAMICA SUL LIBRO

I primi tre capitoli illustrano le caratteristiche di base di questo linguaggio di programmazione. Il Capitolo 1 introduce il QBasic spiegando alcuni concetti fondamentali, il Capitolo 2 mostra come creare ed eseguire un semplice programma, e il Capitolo 3 spiega dettagliatamente l'uso dell'editor. Se si dispone di WordStar, QuickPascal, Turbo Basic o Turbo Pascal si può tralasciare la lettura del Capitolo 3.

Le strutture di programmazione fondamentali e gli enunciati del QBasic sono presentati nei Capitoli 4-6. Benché questi argomenti propri del QBasic vengano trattati molto dettagliatamente, vengono illustrati anche tutti i concetti necessari per la programmazione relativi al BASIC standard.

I Capitoli 7 e 8 forniscono una spiegazione approfondita sui file, la grafica e i suoni. Una buona conoscenza degli argomenti relativi ai file è essenziale, dato che la maggior parte dei dati usati o generati dal computer viene memorizzata su disco. L'importanza della grafica è cresciuta notevolmente a causa delle nuove capacità grafiche introdotte nei personal computer attuali. Il suono, infine, anche se non ricopre un ruolo particolarmente importante, risulta senza dubbio divertente. Nei Capitoli 9 e 10 sono riportate alcune applicazioni che utilizzano intensivamente la grafica e i file.

Gli ultimi due capitoli riportano alcuni programmi di tipo matematico, scientifico e gestionale. Alcune delle applicazioni più estese mostrano come sia possibile suddividere delle lunghe e complesse operazioni di programmazione in piccoli blocchi, che risultano più semplici da leggere e revisionare.

I PROGRAMMI

In genere, un esempio concreto aiuta ad apprendere pienamente un concetto di programmazione. In questo libro, quindi, ogni nuova istruzione sarà accompagnata da un programma, o da un segmento di programma, che ne chiarirà l'uso.

Vengono inoltre presentati i programmi seguenti che non solo dimostrano come sia possibile suddividere una lunga applicazione in più blocchi tra loro collegati, ma svolgono anche delle operazioni utili.

Programma 7.21: Visualizzazione e modifica di un file. Questo programma consente di visualizzare e modificare il contenuto reale di un qualsiasi file. Si possono esaminare e revisionare anche i file binari, come quelli con estensione .EXE.

Programma 9-7: Rappresentazione grafica di una funzione. Questa applicazione consente di rappresentare graficamente delle funzioni con valori molto alti o non definiti. Indipendentemente dal dominio specificato, il programma visualizza una coppia di assi; l'intervallo della scala di valori che appaiono come coordinate y, può essere specificato dall'utente o determinato dal programma.

Programma 9-11: Caratteri personalizzati. Questo programma consente di creare un massimo di 128 nuovi caratteri che possono essere visualizzati in modalità grafica.

Programmi 9-12, 9-13 e 9-14: Gestione di matrici. Queste procedure generiche che sommano, moltiplicano, e invertono delle matrici, possono essere incluse in programmi matematici per svolgere delle operazioni con le matrici.

Programma 10-3: Ammortamento di un prestito. Questo programma consente di determinare il capitale che può essere investito, il numero di mesi necessario per estinguere il debito, o il pagamento mensile richiesto. Dopo aver specificato tutte le informazioni necessarie, il programma visualizza un piano di ammortamento dettagliato.

Programma 10-4: Creazione di un diagramma a torta. Questo programma generico genera una diagramma a torta in grado di rappresentare un massimo di 10 valori. Il programmatore deve soltanto inserire i valori da rappresentare e la loro descrizione negli enunciati DATA. L'applicazione traccerà quindi il grafico aggiungendo le legende corrispondenti.

Programma 10-8: Creazione di un grafico a barre. Questo programma generico rappresenta un massimo di 37 valori definiti dall'utente con un grafico a barre verticali. Il programma utilizza il valore più alto per determinare la scala verticale. Inoltre, delle apposite procedure consentono di effettuare una stampa rapida.

Programma 10-22: Creazione e gestione di un database. Questo programma generico guidato da menu mette a disposizione gli strumenti standard che si possono trovare nella maggior parte dei programmi di database. È possibile creare dei nuovi database specificando i nomi e la lunghezza dei campi, visualizzare il contenuto di un archivio, ordinare i dati, e aggiungere, inserire o modificare dei record. Un unico programma consente di svolgere tutte queste operazioni; non è quindi necessario scrivere un nuovo programma per ogni database che si vuole creare e aggiornare. Si

possono inoltre aggiungere delle nuove funzioni a questo programma generale e averle sempre disponibili per tutti i database che si creano o si utilizzano.

PREREQUISITI

Per leggere questo libro non è necessaria nessuna conoscenza del BASIC. Tuttavia, dato che il QBasic si basa sul BASIC dell'IBM PC, se si conosce già il BASIC IBM, o un linguaggio equivalente come il GW-BASIC, se ne trarrà un sicuro vantaggio. Attualmente, la maggior parte dei programmi scritti con il BASIC IBM possono essere caricati ed eseguiti direttamente in QBasic.

CAPITOLO 1

INTRODUZIONE AL QBASIC

QBasic è la versione del BASIC inclusa nel sistema operativo MS-DOS 5.0. Questo è il linguaggio ideale per i principianti e per i programmatori con poca esperienza. A questo punto, un utente alle prime armi potrebbe porsi diverse domande sul BASIC ed anche sulla programmazione in generale. Questo capitolo risponde ad alcuni dei quesiti più comuni, e fornisce alcune informazioni necessarie per potersi avventurare nel mondo della programmazione. Gli utenti che conoscono già dei linguaggi di programmazione, o che hanno avuto a che fare con le versioni standard del BASIC, possono tranquillamente saltare queste prime sezioni, e reperire alcune informazioni particolarmente interessanti alla fine del capitolo.

INFORMAZIONI SUI COMPUTER E SUI LINGUAGGI DI PROGRAMMAZIONE

Come accennato nella parte introduttiva di questo libro, la versione originale del BASIC, acronimo di Beginner's All-purpose Symbolic Instruction Code, è stata sviluppata al Dartmouth College nel 1964 a scopo didattico. Gli altri linguaggi di programmazione allora esistenti, tra cui l'Assembly, il Fortran e il COBOL, erano molto più difficili da imparare e da utilizzare, e non erano adatti per degli studenti alle prime armi.

Ci si potrebbe a questo punto chiedere che cos'è un linguaggio di programmazione. Molto spesso, le persone che hanno maturato una certa esperienza nel mondo dei personal computer si dimenticano che non tutti capiscono che cosa si intende per linguaggio di programmazione e danno per scontate determinate informazioni. Per assicurarsi di essere tutti sintonizzati sulla stessa lunghezza d'onda, prima di addentrarsi nel BASIC verranno esposti alcuni concetti di base sui personal computer e sui linguaggi.

Un *computer* è uno strumento per eseguire calcoli numerici e per gestire dei simboli in base a una determinata serie di regole. Le regole vengono inviate al computer sotto forma di un *programma*, che consiste semplicemente di una serie di istruzioni e dati che possono essere interpretati dal computer. Ogni riga in un programma viene generalmente denominata *enunciato* e ogni enunciato impartisce normalmente un'istruzione. Si potrebbe sapere che i computer moderni sono sia *digitali* che *binari*. Ciò significa che le informazioni in essi contenute, indipendentemente dal fatto che siano dati o istruzioni, prendono la forma di valori discreti (cifre) che vengono memorizzati in base 2, o notazione binaria (ad esempio, in notazione binaria il numero 5 viene rappresentato come 101). Quindi, ciò che il computer è in grado di interpretare consiste di lunghe stringhe composte da uno e zero, e delle regole fisse stabiliscono come devono essere interpretate le stringhe di codice binario.

La notazione binaria si rivela ideale per il computer, ma certamente non per gli esseri umani. Per indicare al computer di svolgere una determinata operazione, cioè per programmare la macchina, è necessario utilizzare una serie di stringhe composte dai valori uno e zero. Dato che risulta praticamente impossibile per un essere umano utilizzare questa notazione, sono stati sviluppati dei *linguaggi* per facilitare la comunicazione con i computer.

Un linguaggio, indipendentemente dal fatto che si tratti di un linguaggio di programmazione o di comunicazione, consiste di una serie di regole necessarie per costruire degli enunciati, dove un enunciato è un semplice modo per comunicare una parte di un'informazione. Il primo linguaggio per computer è stato denominato *Assembly*, utilizzava un codice di due lettere per indicare i possibili comandi disponibili, e consentiva di rappresentare i numeri nella notazione convenzionale, quella in base 10. Prima di poter caricare un programma in memoria ed eseguirlo, lo si doveva convertire in notazione binaria tramite un apposito programma denominato *assembler*. Dato che anche l'Assembly si è dimostrato di difficile utilizzo, sono stati sviluppati altri linguaggi in cui i codici composti da due lettere sono stati sostituiti da intere parole. Come menzionato in precedenza, il Fortran e il COBOL sono due esempi di questi tipi di linguaggio.

Per venire incontro alle esigenze delle scuole, che richiedevano un linguaggio più adatto per gli studenti che si apprestavano ad affrontare la programmazione, è stato sviluppato il BASIC, un linguaggio più semplice, ma altrettanto efficace, che consentiva di eseguire i programmi eliminando alcune operazioni normalmente necessarie.

Nato nel 1964 a Dartmouth, il BASIC ha conosciuto un notevole sviluppo negli anni successivi, ed ha subito diverse modifiche e miglioramenti. Nel 1978 è stato adottato uno standard che ha ridotto al minimo i requisiti per il BASIC. In questo libro, si farà riferimento a questo linguaggio come BASIC standard.

CONCETTI FONDAMENTALI SUL PC BASIC

BASICA è la versione avanzata del BASIC scritto dalla Microsoft per i personal computer IBM ed incluso nelle versioni del DOS precedenti alla 5.0. Per i computer IBM compatibili è disponibile una versione equivalente, il GW-BASIC. BASICA mette a disposizione circa 190 comandi ed offre più possibilità del BASIC standard.

USO DI UN INTERPRETE

Nella sezione precedente è stato detto che il BASIC consente di eseguire un programma senza dover effettuare alcune operazioni che risultano necessarie quando si utilizzano altri linguaggi. Come si è visto in precedenza, un computer è in grado di eseguire un programma se le istruzioni in esso contenute sono espresse sotto forma di stringhe composte da valori uno e zero.
Quando si scrive un programma in un qualsiasi linguaggio diverso dal BASIC, gli enunciati devono essere convertiti in linguaggio macchina da un altro programma prima di poter essere eseguiti. Nella maggior parte dei casi, l'intero programma viene convertito prima di essere eseguito. Ciò non accade con il BASIC, poiché questo linguaggio viene *interpretato*.

Un *interprete* è un programma, con istruzioni in linguaggio macchina, scritto per 'capire' ed eseguire delle istruzioni di un altro linguaggio, come ad esempio il BASIC. Quando un computer esegue un programma in BASIC standard, esegue in realtà un altro programma che legge, capisce, ed impartisce correttamente ciascuna istruzione del programma in BASIC, un enunciato alla volta. Un linguaggio eseguito in questo modo viene chiamato *linguaggio interpretato.*

Uno svantaggio che deriva dall'uso di un linguaggio interpretato è che il computer deve leggere e convertire un'istruzione ogni volta che viene incontrata, anche se lo stesso enunciato appare 1000 volte in un unico ciclo. Queste possibili ripetizioni comportano un notevole spreco di tempo che non può essere considerato un problema reale per uno studente che sta apprendendo l'uso del personal computer, ma che risulta improponibile per un programmatore che deve sviluppare una grossa applicazione.

Con l'avvento del QBasic le cose sono cambiate. Analogamente a linguaggi come il Fortran, il COBOL, il Pascal e il C, QBasic viene *compilato.* Ciò significa che l'intero

programma viene convertito in linguaggio macchina prima di essere eseguito. Il programma che esegue questa conversione viene chiamato *compilatore*.

Il compilatore del QBasic è un programma che legge e capisce le istruzioni di un programma BASIC, verifica la presenza di alcuni tipi di errore, e converte tutti gli enunciati in linguaggio macchina. Un compilatore risulta più efficiente di un interprete, dato che ciascuna istruzione deve essere letta e convertita una sola volta. Inoltre, il compilatore non impartisce le istruzioni che converte, ma produce solamente una serie di istruzioni in linguaggio macchina che possono essere eseguite direttamente dal computer per svolgere le operazioni del programma BASIC. Due delle caratteristiche più importanti del QBasic sono la velocità e l'efficienza con cui vengono compilati i programmi.

ALCUNE CONSIDERAZIONI

Tralasciando la differenza che intercorre tra un linguaggio interpretato ed uno compilato, il BASIC ha sempre offerto più vantaggi degli altri linguaggi, come pure il BASICA e il QBasic.

Sia il BASIC standard che il BASICA sono facili da imparare, dato che le istruzioni vengono scritte utilizzando parole familiari in lingua inglese. Le nuove *variabili*(delle posizioni di memoria cui viene assegnato un nome) possono essere aggiunte in qualsiasi momento, a differenza degli altri linguaggi in cui le variabili devono essere definite, o *dichiarate*, all'inizio del programma, e ci sono poche strutture complesse. BASICA mette a disposizione dei comandi per la gestione della grafica, del suono, e per il rilevamento di eventi particolari.

Dato che i programmi vengono interpretati, possono essere collaudati e verificati interrompendo l'esecuzione in qualsiasi momento ed analizzando il contenuto delle variabili.

Insieme a questi vantaggi, tuttavia, ci sono delle limitazioni. Innanzitutto, in entrambe le versioni del BASIC è necessario identificare ciascun enunciato con un *numero di riga*. Questi numeri indicano all'interprete BASIC la sequenza di esecuzione dei comandi, e servono inoltre come indirizzi per spostare l'esecuzione del programma in un altro punto.

Sfortunatamente, in alcuni casi i numeri di riga rendono più complicata la stesura di un programma, e risulta più difficoltoso tener traccia delle *subroutine*, dei piccoli programmi contenuti all'interno del programma principale che svolgono operazioni specifiche

In secondo luogo, tutte le variabili del BASIC sono *globali*; ciò significa che mantengono il loro valore in tutte le parti del programma. Ciò rende più difficile il

riutilizzo delle subroutine, dato che queste devono avere nomi di variabili unici per evitare un'interferenza con le altre presenti nel programma. Anche all'interno del programma principale bisogna prestare attenzione a non utilizzare la stessa variabile in due contesti differenti.

In terzo luogo, l'enunciato IF...THEN, che consente di determinare il modo in cui procedere sulla base di condizioni specificate, non è molto flessibile e chiaro. Altri linguaggi dispongono di numerosi comandi che consentono di 'prendere delle decisioni'; questi enunciati di decisione vengono chiamati *strutture di controllo*. Non si esagera dicendo che le strutture di controllo sono il cuore della programmazione, e che una grande varietà di istruzioni di questo tipo e una notevole flessibilità aumentano notevolmente le potenzialità di un linguaggio di programmazione.

Infine, il BASIC standard e il BASICA sono confinati nel limite di 64K di memoria, che deve servire sia come spazio di lavoro che come area di memorizzazione delle variabili. Nel 1981, 64K costituivano una grossa quantità di memoria. Al giorno d'oggi, invece, con l'avvento della grafica VGA e dei nuovi microprocessori, 64K sono veramente scarsi. Non bisogna inoltre sottovalutare la lentezza causata dall'interpretazione dei programmi, che risulta particolarmente fastidiosa quando si scrivono grosse applicazioni.

ALLA SCOPERTA DI QBASIC

Come si potrà immaginare, QBasic ha superato tutte queste limitazioni pur conservando i vantaggi delle prime versioni del BASIC.

Tutti i programmatori sono concordi nell'affermare che un moderno linguaggio di programmazione non dovrebbe richiedere i numeri di riga, ma dovrebbe disporre di numerose strutture di controllo, di *variabili locali* (cioè delle variabili che si applicano ad una sola porzione di un programma), e di funzioni che consentano di passare e ricevere dei valori dalle procedure. Una *procedura* è una porzione di un programma che esegue un'operazione specifica; è in pratica l'equivalente di una subroutine. Tutte queste caratteristiche sono disponibili in QBasic. Per la maggior parte dei programmi, QBasic può accedere all'intera memoria convenzionale disponibile nel computer, invece che ai soli 64K menzionati in precedenza. Inoltre, dato che QBasic è un linguaggio compilato, i programmi vengono eseguiti più rapidamente. Bisogna inoltre considerare il fatto che è molto semplice sfruttare tutte queste potenzialità, anche per un utente inesperto. Nel prossimo capitolo si vedrà come creare un piccolo programma utilizzando l'*editor* del QBasic.

La sezione seguente riporta alcune informazioni che potrebbero risultare utili agli utenti familiari con le versioni precedenti del BASIC. Se non è mai stato utilizzato il linguaggio BASIC, si può passare direttamente all'inizio del Capitolo 2.

UTENTI ESPERTI: INTRODUZIONE AL QBASIC

Se già si conosce il BASIC standard, si può iniziare a programmare in QBasic senza sforzi particolari. Per poter scrivere dei piccoli programmi, è necessaria circa un'ora per apprendere i concetti essenziali relativi all'inserimento, all'esecuzione, e alla modifica dei programmi. Dopo ciò si può leggere questo libro per esplorare le nuove possibilità messe a disposizione dal QBasic.

Ci si potrebbe innanzitutto chiedere che cosa accade con i numeri di riga. In QBasic, i numeri di riga sono opzionali; questi non sono necessari per tenere traccia della sequenza di esecuzione dei comandi, ma vengono principalmente usati per gli enunciati GOTO e GOSUB. Tuttavia, la disponibilità di procedure e di numerose strutture di controllo rendono queste istruzioni alquanto obsolete (per anni, i docenti hanno minacciato di bocciare immediatamente qualsiasi studente che avesse usato un comando GOTO in un programma).

Per indicare la destinazione ad un comando GOTO o GOSUB, si può utilizzare un'*etichetta* (label) al posto del numero di riga. Anche se l'abitudine fa pensare che i numeri di riga siano insostituibili, con il tempo ci si renderà conto che le etichette rendono la struttura di un programma molto più flessibile.

Sebbene il QBasic possa fare a meno dei numeri di riga e debba compilare un programma prima di eseguirlo, è per molti aspetti compatibile con le versioni precedenti del BASIC. Si può caricare ed eseguire in QBasic la maggior parte dei programmi scritti in BASICA, in BASIC, e in GW-BASIC. Sono sufficienti delle piccole modifiche agli enunciati DRAW, PLAY, CLEAR e CHAIN. Si noti; tuttavia, che i comandi che agiscono sul programma stesso, come LIST, EDIT, RENUM e AUTO, non possono essere utilizzati in QBasic.

Un altro aspetto da considerare se è già stata utilizzata una versione precedente del BASIC, sono le differenze che intercorrono tra un linguaggio interpretato ed uno compilato. Ad esempio, in un linguaggio interpretato i nomi di variabile molto lunghi occupano più memoria di quelli corti, mentre in un linguaggio compilato tutti i nomi di variabile utilizzano la stessa quantità di memoria. Quindi, in QBasic si possono utilizzare dei nomi descrittivi senza preoccuparsi dello spazio da essi richiesto. Si faccia tuttavia attenzione a non esagerare, poiché risulta tedioso digitare numerose volte un lungo nome di variabile.

Si tenga inoltre presente che il BASIC standard rileva la presenza di eventuali errori quando il programma è in esecuzione, e che alcuni errori si manifestano solamente dopo un test intensivo. Inoltre, in certe condizioni alcuni tipi di errore non si possono verificare. L'editor e il compilatore del QBasic, invece, analizzano l'intero programma prima dell'esecuzione, rilevando molti tipi di errore. In questo modo si risparmia del tempo e non si corre il rischio di perdere dei giorni per capire l'origine di un problema.

QBasic è un linguaggio ad alto livello e costituisce un'ottima introduzione alla *programmazione strutturata.* Questo tipo di programmazione consente di progettare un programma facile da scrivere, da leggere, da revisionare, e da collaudare. Con questa strategia, i problemi vengono suddivisi in più categorie e vengono sviluppate diverse porzioni di programma per ogni specifica operazione. In questo modo viene generato un programma composto da piccole porzioni tra loro indipendenti, che possono essere riutilizzate in altri programmi. La programmazione strutturata richiede l'uso di strutture di controllo e di procedure, entrambe disponibili in QBasic.

Le operazioni fondamentali necessarie per scrivere ed eseguire un programma in QBasic, possono essere riassunte nel modo seguente:

1. avviare QBasic digitando `QBASIC` al prompt del DOS e premendo Invio;
2. premere il tasto Esc per rimuovere la schermata iniziale;
3. digitare il programma come se si stesse lavorando in un programma di elaborazione testi;
4. una volta terminata la digitazione, premere la combinazione di tasti Maiusc-F5 per eseguire il programma.

Questo è tutto ciò che bisogna fare. Nel prossimo capitolo verranno fornite informazioni più dettagliate.

CONCLUSIONE

Se ci si appresta ad utilizzare seriamente il QBasic, si devono considerare due strumenti avanzati disponibili per i programmatori: QuickBASIC e BASIC PDS.

QuickBASIC è una versione avanzata di QBasic, mentre BASIC PDS è una versione avanzata ed estesa di QuickBASIC. La caratteristica più importante di QuickBASIC è la possibilità di compilare un programma in un formato eseguibile direttamente dal DOS, proprio come accade per le applicazioni gestionali come Lotus 1-2-3, Microsoft Word, e dBASE. QuickBASIC dispone inoltre di 10 comandi addizionali e di ulteriori funzioni di collaudo. BASIC PDS, invece, aggiunge nuove funzioni per la gestione dei database ed altre caratteristiche molto importanti per la programmazione avanzata. Qualsiasi programma scritto in QBasic può essere eseguito in QuickBASIC o BASIC PDS.

A questo punto, dopo una breve panoramica sul QBasic, si imparerà a scrivere un programma utilizzando l'apposito editor.

CAPITOLO 2

UNA BREVE PANORAMICA SU QBASIC

Prima di poter avviare un programma, eseguendo in sequenza ciascuna istruzione, è necessario inserire il programma nel computer. Ovviamente, si utilizzerà la tastiera, e forse anche il mouse, per svolgere questa operazione. Il testo, tuttavia, non viene inserito direttamente in QBasic, ma bisogna prima utilizzare l'*editor di QBasic.*

L'editor di QBasic è una specie di programma di elaborazione testi. Esempi di programmi di questo genere sono Microsoft Word, WordStar, e WordPerfect. Coloro che non sono familiari con questi programmi, o non conoscono il termine elaborazione testi (o word processor), devono semplicemente sapere che un programma di questo tipo è uno strumento che consente di digitare, visualizzare, e ridisporre un testo nel PC.

Un programma di elaborazione testi conferisce al computer maggiori potenzialità rispetto ad una macchina da scrivere; tra le altre cose, si possono correggere facilmente eventuali errori di battitura, e inserire delle nuove parole a metà di un documento senza dover ridigitare delle singole righe o delle intere pagine. L'editor di QBasic mette a disposizione queste funzioni per facilitare la stesura di un programma.

Questo capitolo presenta le informazioni essenziali necessarie per scrivere, revisionare ed eseguire un programma utilizzando QBasic. Dato che l'editor è il programma che

verrà utilizzato per svolgere queste operazioni, questo capitolo introduce le funzioni principali dell'editor di QBasic. Il Capitolo 3 presenterà ulteriori informazioni sull'editor, trattando delle funzioni più avanzate. Gli utenti che conoscono WordStar, o altri editor orientati alla programmazione, possono tralasciare la lettura del Capitolo 3. Per editor orientati alla programmazione si intende un programma di elaborazione testi contenuto all'interno di un altro programma. Gli esempi più rilevanti sono gli editor contenuti nella maggior parte degli altri linguaggi di programmazione, come il Pascal e il C. Se si è in grado di utilizzare un editor di un altro linguaggio di programmazione, quello del QBasic non dovrebbe creare nessun problema.

I PRIMI PASSI

Si installi il DOS 5.0, o una versione successiva, nel disco fisso del computer seguendo le istruzioni riportate nei manuali del DOS. I due file necessari per utilizzare QBasic sono QBASIC.EXE e QBASIC.HLP. Se si pensa di utilizzare la grafica e si dispone di una scheda Hercules, si deve copiare anche il file MSHERC.COM. Se si utilizza QBasic su disco fisso, si potrebbero copiare questi file nella directory DOS. In caso contrario, si dovrebbe prima eseguire una copia dei dischetti originali e lavorare solamente sulle copie.

Si avvii QBasic dal disco. Se non si conosce la procedura necessaria per avviare il programma, si faccia riferimento all'Appendice H per le informazioni del caso. Dopo aver avviato QBasic ed aver rimosso la schermata di apertura tramite la pressione del tasto Esc, apparirà la schermata principale riportata nella Figura 2.1.

LA FINESTRA PRINCIPALE

Si noti che la barra dei menu nella parte superiore dello schermo contiene otto voci. Un menu, naturalmente, è un semplice elenco di comandi disponibili che possono essere selezionati. Le otto voci di menu presenti sulla barra sono le seguenti:

File Mostra un menu a tendina contenente dei comandi relativi alle operazioni sui file, come il caricamento e il salvataggio di un programma. Per *file* si intende una singola serie di dati continua. Ogni programma scritto in QBasic viene memorizzato in un file separato.

Modifica Mostra un menu a tendina contenente dei comandi relativi alle operazioni di revisione che consentono, ad esempio, di spostare del testo da una parte a un'altra di un programma.

Visualizza Mostra un menu a tendina con dei comandi che consentono di visualizzare delle schermate specifiche di un programma.

Cerca Mostra un menu a tendina che contiene i comandi standard di ricerca e sostituzione che si trovano normalmente nei programmi di

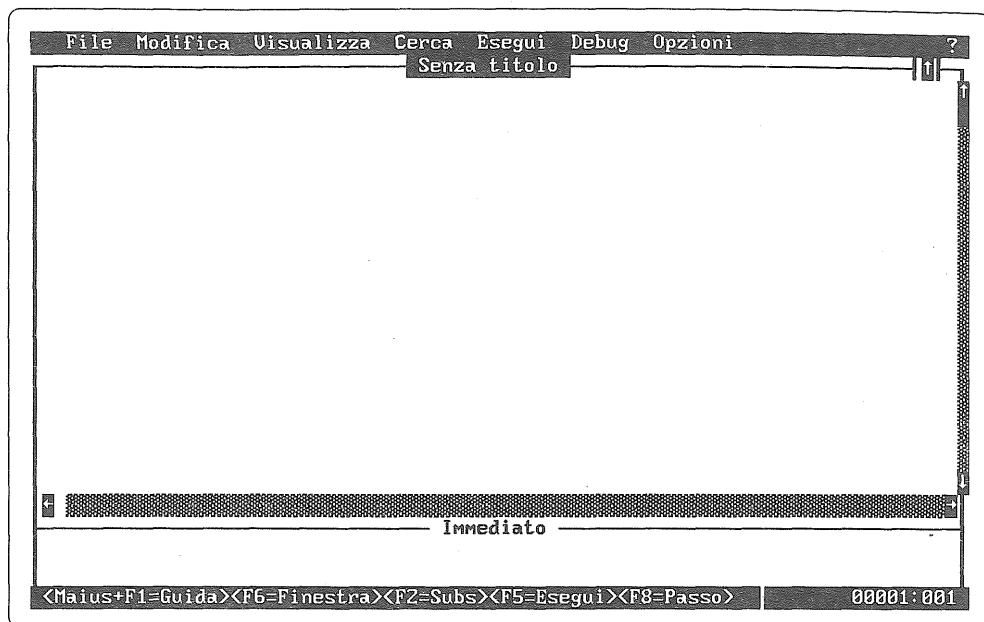



Figura 2.1 *La schermata principale*

elaborazione testi. Grazie a questi comandi, si può localizzare rapidamente una parola o una frase particolare all'interno del programma, ed eventualmente sostituirla con dell'altro testo specificato.

Esegui Mostra un menu a tendina che contiene dei comandi che agiscono sull'esecuzione dei programmi.

Debug Mostra un menu a tendina con dei comandi che consentono di collaudare un programma rilevando gli eventuali presenti.

Opzioni Mostra un menu a tendina che contiene dei comandi che consentono di personalizzare QBasic. Per esempio, si possono modificare i colori dello schermo.

? Mostra un menu a tendina contenente dei comandi che consentono di accedere al sistema di aiuto in linea.

Nel momento in cui si evidenzia uno di questi comandi nella barra dei menu, come ad esempio File nella Figura 2.2, QBasic attende che l'utente scelga un comando del menu. Per attivare la barra dei menu, si prema il tasto Alt. Si utilizzino quindi i tasti freccia, a volte chiamati tasti cursore, per spostare l'evidenziatore sulle varie voci. Per visualizzare i comandi contenuti in un menu, si prema il tasto Invio.

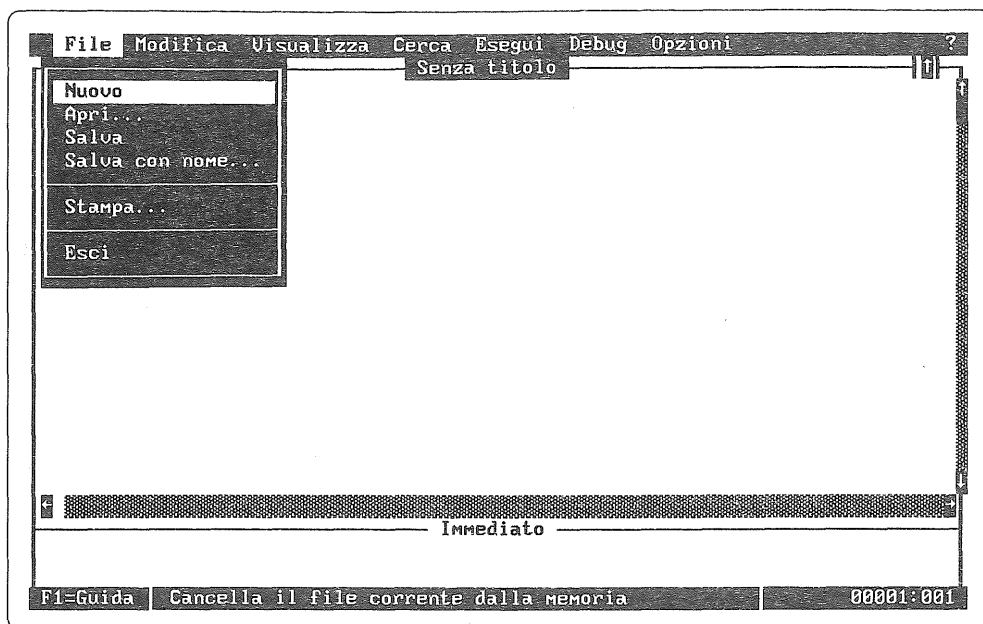



Figura 2.2 *Il menu File*

Le altre componenti dello schermo sono le seguenti:

Barra del titolo La barra del titolo nella parte superiore dello schermo visualizza le parole Senza titolo fino a quando non si salva e si assegna un nome al proprio lavoro. I comandi per salvare i programmi si trovano nel menu File.

Finestra di visualizzazione Tutte le operazioni di inserimento e modifica del testo di un programma vengono effettuate in questa finestra.

Finestra Immediato Si prema il tasto F6 per attivare questa finestra. È possibile digitare nella finestra Immediato un comando di QBasic ed eseguirlo da solo immediatamente.

Barra di stato La barra di stato mostra quali tasti possono essere correntemente utilizzati, la posizione del cursore, e delle informazioni sui comandi di menu. Il cursore indica la posizione in cui sarà inserito un carattere nel momento in cui verrà digitato da tastiera, ed appare come un piccolo quadrato (o una piccola linea) lampeggiante, a seconda dell'impostazione del sistema.

I MENU DI QBASIC

Dopo aver avviato QBasic, viene attivata la finestra di visualizzazione. Tramite il tasto Alt si accede alla barra dei menu, da cui è possibile selezionare la voce desiderata. Ci sono due metodi per scegliere un comando dalla barra dei menu:

premendo la prima lettera del comando desiderato, utilizzando indifferentemente le lettere maiuscole o minuscole;

utilizzando i tasti freccia per spostare il rettangolo evidenziato sulla voce desiderata e premendo il tasto Invio.

Ad esempio, se si preme la lettera F dopo aver attivato la barra dei menu, viene aperto il menu File (si veda la Figura 2.2). Questo menu viene chiamato menu a tendina. A questo punto, premendo la lettera E viene selezionato il comando Esci che chiude la sessione di lavoro di QBasic e ritorna al sistema operativo. Segue un altro esempio di utilizzo dei menu. Si attivi la barra dei menu premendo il tasto Alt, e si prema quindi il tasto Destra fino ad evidenziare il comando ?. Si prema Invio per aprire il menu a tendina corrispondente riportato nella Figura 2.3. A questo punto, si può utilizzare il tasto Basso per evidenziare il comando desiderato e premere Ritorno per impartirlo.

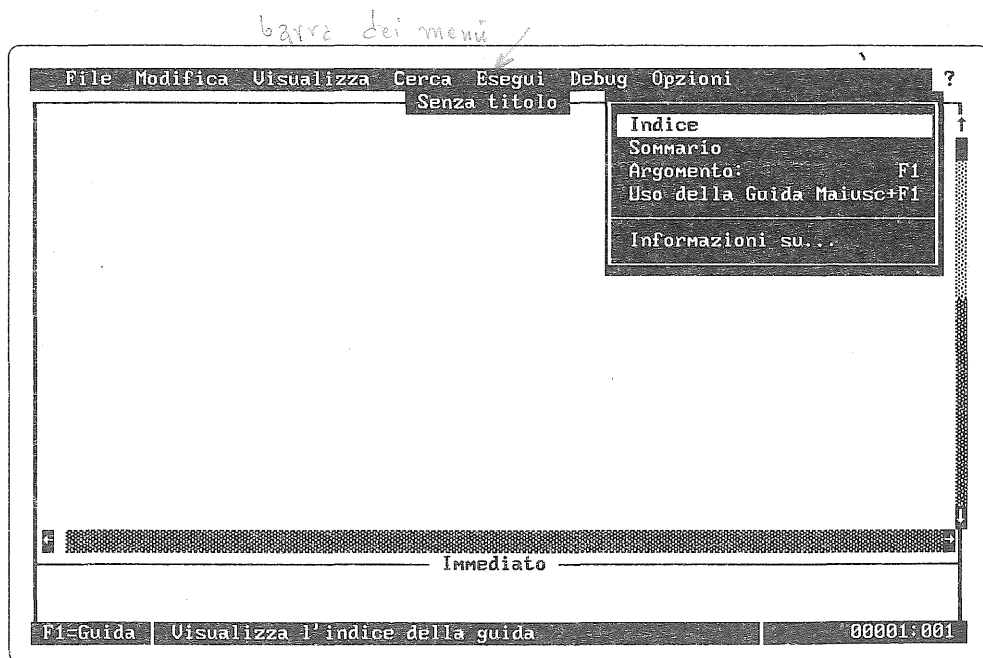



Figura 2.3 *Il menu ?*

Nella barra di stato appaiono delle informazioni relative al comando correntemente evidenziato. Come si può notare, ad alcuni comandi sono associati dei tasti scorciatoia che vengono utilizzati per impartire il comando loro associato direttamente dalla finestra di visualizzazione, senza prima attivare la barra dei menu. Per ritornare alla finestra di visualizzazione si deve premere il tasto Esc. Se questa è la prima volta che si utilizza QBasic, si potrebbero voler esaminare altri comandi dei menu a tendina del programma. A questo scopo, si faccia riferimento all'Appendice E che riporta una spiegazione dei vari comandi disponibili.

CREAZIONE ED ESECUZIONE DI UN PROGRAMMA

Che ci si creda o meno, si è ora pronti per scrivere il primo programma in QBasic. Il testo del programma deve essere digitato nella finestra di visualizzazione. Questa finestra è attiva quando il cursore lampeggiante appare al suo interno (per finestra attiva si intende la finestra in cui vengono inseriti i caratteri digitati). Se necessario, si prema Esc, oppure il tasto F6, fino a quando il cursore si sposta nella finestra di visualizzazione.

L'editor di QBasic mette a disposizione diverse funzioni per facilitare la stesura di un programma. Al momento, tuttavia, sarà sufficiente seguire queste direttive:

- digitare ciascuna riga del programma come se si stesse utilizzando una macchina da scrivere. Dopo aver digitato completamente una riga, si prema il tasto Invio;
- utilizzare i tasti cursore (Alto, Basso, Destra e Sinistra) per spostarsi in un punto qualsiasi del programma;
- per cancellare un carattere, premere il tasto Backspace (posizionato sopra al tasto Invio) o Canc per eliminare, rispettivamente, il carattere che si trova a sinistra o alla posizione del cursore. I caratteri che si trovano a destra di quelli cancellati vengono automaticamente spostati verso sinistra per riempire lo spazio lasciato vuoto dall'operazione di cancellazione;
- per inserire un carattere, posizionare il cursore nel punto desiderato e digitare la lettera appropriata. I caratteri a destra del cursore vengono automaticamente spostati verso destra per accomodare il nuovo inserimento.
- per cancellare un'intera riga, si posizioni il cursore su un carattere qualsiasi della riga da rimuovere. Si tenga quindi premuto il tasto Ctrl (Ctrl indica il tasto Control) e si prema la lettera Y. Questa combinazione di tasti viene identificata come Ctrl-Y.

Con queste informazioni si può ora procedere all'inserimento del programma.

STESURA DI UN PROGRAMMA

L'esercizio seguente illustra la procedura di base che deve essere seguita per creare ed eseguire un programma. I vari comandi utilizzati in questo esercizio verranno spiegati in seguito. Al momento, si seguano solamente le istruzioni riportate e si osservino i risultati.

1. Se necessario, utilizzare i tasti cursore e la combinazione Ctrl-Y per cancellare il contenuto della finestra di visualizzazione;
2. digitare la parola

   ```
   cls
   ```

   in lettere minuscole e premere il tasto Invio. Si noti che la parola viene convertita automaticamente in lettere maiuscole.
   L'istruzione CLS indica a QBasic di cancellare il contenuto dello schermo. CLS è un esempio di *parola chiave*, o *parola riservata*. Le parole chiave hanno un significato speciale in QBasic, e non possono essere utilizzate come nomi di variabili all'interno del programma. Ci sono circa 200 parole riservate in QBasic.
   Un *enunciato*, o istruzione, consiste di una parola chiave ed eventualmente di alcuni parametri. I parametri sono delle parole o dei valori specificati dall'utente che servono per controllare la funzione dell'istruzione. Ogni enunciato deve essere inserito in una riga separata, ed indica al computer di eseguire una determinata operazione. Un programma è composto da una serie di enunciati;
3. nella seconda riga della finestra di visualizzazione digitare:

   ```
   FOR i=1 TO
   ```

   e premere Invio. Il numero 1 nella riga sopra riportata deve essere digitato premendo il tasto 'uno' che si trova in alto a sinistra del tasto Q, e *non* il tasto l (elle).
   Appare a questo punto la finestra riportata nella Figura 2.4 ad indicare che è stato commesso un errore nella digitazione del comando. Questa finestra viene normalmente chiamata finestra di *dialogo*. Le finestre di dialogo vengono visualizzate in diverse situazioni, sia per segnalare degli errori che per richiedere delle informazioni. Per spostarsi tra le diverse opzioni nella finestra di dialogo si utilizzi il tasto Tab, mentre per rimuovere dallo schermo la finestra si prema Esc.
   Premere il tasto Esc per rimuovere la finestra di dialogo e continuare la digitazione sulla stessa riga in modo da ottenere

   ```
   FOR i = 1 TO 100
   ```

   Premere il tasto Invio. Si noti che sono stati aggiunti degli spazi prima e dopo il segno uguale per rendere la riga più leggibile;

4. digitare gli altri enunciati presenti nel listato sotto riportato. Questo è il primo esempio di programma in QBasic. Questo programma contiene una serie di istruzioni che indicano al computer di visualizzare per 100 volte la parola 'Ciao' dopo aver cancellato il contenuto dello schermo e, come ogni programma QBasic, termina con l'istruzione END. Si dice che il computer esegue un programma quando vengono impartite, una per una, tutte le istruzioni in esso contenute;

```
CLS
FOR i = 1 TO 100
PRINT "Ciao"
NEXT i
END
```

5. per eseguire un programma, si tenga premuto il tasto Maiusc e si prema F5. La visualizzazione cambia e viene mostrato lo schermo output;
6. premere un tasto qualsiasi per ritornare alla finestra di visualizzazione, dove si potrà continuare con la modifica del programma;
7. premere il tasto F4 per visualizzare nuovamente lo schermo output, e premerlo nuovamente per ritornare alla finestra di visualizzazione. Il Tasto F4 funge da *interruttore* e consente di passare tra le due finestre;
8. se si desidera, si può eseguire nuovamente il programma premendo la combinazione di tasti Maiusc-F5;
9. se sono stati commessi degli errori nelle istruzioni digitate, QBasic li rileva quando si esegue il programma tramite la pressione dei tasti Maiusc-F5. Per vedere un esempio di questo tipo, si ritorni alla finestra di visualizzazione, si sposti il cursore sulla quarta riga del programma, si cancelli la lettera X dal comando NEXT, e si esegua il programma. Questa volta, appare una finestra di dialogo che indica la presenza di un errore di sintassi; la riga di programma 'incriminata' viene evidenziata.
   Si selezioni Guida per ottenere delle informazioni sul tipo di errore verificatosi, oppure Esc per ritornare alla finestra di visualizzazione. Il cursore viene

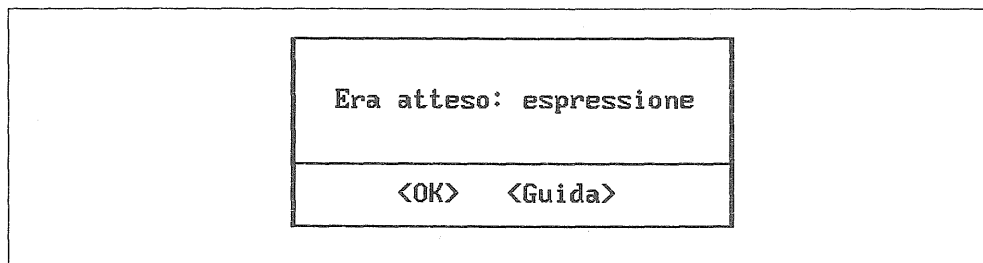



Figura 2.4 *Una finestra di dialogo*

posizionato automaticamente sulla riga da correggere. Una volta corretto l'errore, si possono utilizzare i tasti cursore per spostarsi in qualunque punto del programma;

10. si modifichi il programma aggiungendo l'enunciato

    ```
    PRINT i;
    ```

    prima della terza riga.
    Per creare lo spazio necessario per inserire la nuova istruzione, si sposti il cursore alla fine della seconda riga e si prema Invio. In alternativa, si può spostare il cursore all'inizio della terza riga, premere Invio e spostare il cursore in alto di una riga. Si digiti la nuova istruzione. Nel momento in cui si sposta il cursore dalla riga corrente, QBasic rileva la presenza di alcuni tipi di errore, converte le parole chiave in lettere maiuscole, e modifica la spaziatura;

11. si esegua il nuovo programma. Si può notare che ogni parola Ciao è ora numerata;

12. premere un tasto qualsiasi per ritornare alla finestra di visualizzazione;

13. si può a questo punto *memorizzare* il programma in un dischetto o nel disco fisso, in modo da poterlo eseguire in una sessione di lavoro futura senza doverlo ridigitare. A questo scopo, si attivi la barra dei menu premendo il tasto Alt, si prema la lettera F per selezionare il menu File, e quindi la lettera V per impartire il comando Salva con nome. Nel corso di questo libro, combinazioni di tasti di questo tipo verranno indicate come Alt/F/V.
    Appare una finestra di dialogo che richiede il nome da assegnare al programma. Si digiti un nome composto al massimo da otto lettere e/o numeri e si prema Invio. Per esempio, si potrebbe utilizzare il nome MIOPROG. Il programma viene salvato su disco, nella directory da cui è stato avviato QBasic, nel file denominato MIOPROG.BAS. Le lettere BAS costituiscono l'estensione del file e servono ad indicarne il tipo;

14. si supponga ora di voler scrivere un nuovo programma. Per rimuovere MIOPROG dalla finestra di visualizzazione, si attivi la barra dei menu, si selezioni il menu File, e si scelga il comando Nuovo. Se si preme Alt/F/N prima di salvare un programma che ha subito nelle modifiche, appare una finestra di dialogo che richiede se si vuole salvare o meno il file prima di rimuoverlo dalla memoria. Si utilizzi il tasto Tab per evidenziare l'opzione desiderata, e si prema quindi Invio;

15. in qualsiasi momento si può ricaricare il programma MIOPROG nella finestra di visualizzazione premendo i tasti Alt/F/A (Apri), digitando MIOPROG alla posizione del cursore, e premendo quindi il tasto Invio;

16. per uscire da QBasic, si prema semplicemente Alt/F/E (Esci). Se sono state apportate delle modifiche nel programma corrente dopo l'ultimo salvataggio, QBasic chiede se si desidera salvare il file prima di uscire dal programma.

Come si può vedere, la creazione, la modifica, l'esecuzione e il salvataggio di un programma sono operazioni molto semplici. Nel prossimo capitolo verranno fornite ulteriori spiegazioni sulle varie funzioni dell'editor, prima di esaminare i comandi del linguaggio di programmazione.

Prima di passare al prossimo capitolo, gli utenti che conoscono il BASIC standard dovrebbero leggere la sezione seguente che riporta alcuni comandi di questo linguaggio ed il loro equivalente in QBASIC. Gli utenti che invece non hanno mai utilizzato il BASIC, possono tralasciare questa sezione e passare direttamente al Capitolo 3.

UTENTI ESPERTI: COME ESEGUIRE ALCUNE OPERAZIONI DEL BASIC STANDARD

Segue un elenco di alcuni dei comandi 'diretti' più comuni del BASIC standard. Per diretti si intendono quei comandi che il BASIC esegue immediatamente non appena vengono digitati e viene premuto il tasto Invio. Per ognuno di questi comandi vengono fornite le spiegazioni per eseguire le stesse operazioni in QBasic. Alcune di queste operazioni possono essere eseguite anche fuori dall'editor, come spiegato nell'Appendice E.

LIST Nel BASIC standard, questo comando visualizza l'intero contenuto del programma corrente. In QBasic, invece, una porzione del programma corrente è sempre visibile nella finestra di visualizzazione. Per esaminare altre porzioni del programma, si possono utilizzare i tasti PgUp, PgDn, Ctrl-PgUp, Ctrl-PgDn e i tasti cursore.

SAVE Equivale al comando Alt/F/S. Se non è ancora stato assegnato un nome al programma, QBasic mostra una finestra di dialogo per richiedere un nome.

LLIST Invia tutto il listato del programma corrente alla stampante di default, ed equivale al comando Alt/F/M.

LOAD Si prema Alt/F/A per impartire il comando Apri. Questa sequenza di tasti richiama la finestra di dialogo Apri che visualizza il filtro *.BAS nella casella Nome del file. A questo punto ci sono due opzioni: si può caricare un programma in memoria digitandone il nome nella casella Nome del file e premendo Invio, oppure richiedere un elenco dei file presenti nel disco utilizzando un apposito filtro. A questo scopo, si possono utilizzare i metacaratteri del DOS * e ?; il filtro di default, *.BAS, visualizza tutti i file con un qualsiasi nome e con estensione .BAS. Si può quindi selezionare il programma desiderato dall'elenco utilizzando il tasto Tab, i tasti cursore e il tasto Invio.

NEW Questo comando serve per rimuovere il programma corrente dalla memoria

ed equivale alla sequenza di tasti Alt/F/N. Se sono state apportate delle modifiche al programma corrente, QBasic chiede conferma dell'operazione.

SYSTEM Consente di ritornare al prompt del DOS ed equivale alla sequenza di tasti Alt/F/E. Se sono state apportate delle modifiche al programma corrente, QBasic chiede conferma dell'operazione.

In questo capitolo sono state presentate le informazioni di base necessarie per creare ed eseguire un piccolo programma in QBasic utilizzando l'editor. Nel prossimo capitolo verranno esaminate altre funzioni dell'editor. Se si conosce WordStar o un altro editor incorporato in un linguaggio di programmazione, si può tralasciare la lettura del Capitolo 3. Il Capitolo 4 tratta il linguaggio vero e proprio e spiega come utilizzare i comandi per la gestione dei numeri e dei caratteri.

CAPITOLO 3

L'EDITOR DI QBASIC

Questo capitolo si rivolge agli utenti che vogliono conoscere l'editor di QBasic in modo più approfondito, prima di addentrarsi nel linguaggio di programmazione. Come già detto in precedenza, se si conosce WordStar, o gli editor inclusi in QuickPascal, Turbo Basic o Turbo Pascal, si è già in possesso delle nozioni necessarie per procedere con il capitolo successivo. Se si desidera, si può leggere rapidamente questo capitolo per rinfrescarsi la memoria e per vedere le funzioni specifiche dell'editor di QBasic.

Gli utenti meno esperti, soprattutto quelli alle prime esperienze con il personal computer, dovrebbero leggere interamente questo capitolo che fornisce una spiegazione esauriente dell'editor di QBasic. Una volta apprese le funzioni qui esposte, si sarà in grado di iniziare a lavorare con QBasic.

CONSIDERAZIONI

I programmi vengono digitati da tastiera come se si stesse utilizzando una macchina da scrivere. In gergo informatico, la procedura necessaria per scrivere un programma viene chiamata *editing*, da cui deriva il nome editor, il programma che consente di scrivere e modificare un programma in QBasic.

FUNZIONI DELL'EDITOR DI QBASIC

Prima di procedere, ci si soffermerà sulle funzioni disponibili nell'editor. Le operazioni principali che si possono svolgere con l'editor di QBasic sono analoghe a quelle della maggior parte dei programmi di elaborazione testi. Come detto nel Capitolo 2, un programma di elaborazione testi consente di inserire e gestire del testo.

Spostamento del cursore Si può spostare il cursore in un punto qualsiasi del programma. I tasti freccia del tastierino numerico spostano il cursore di una posizione nella direzione indicata della freccia. Se usati in combinazione con il tasto Ctrl, i tasti Sinistra e Destra spostano il cursore, rispettivamente, all'inizio della parola a sinistra e a destra della posizione corrente del cursore. Inoltre, l'editor dispone di comandi che permettono di spostare il cursore all'inizio o alla fine della riga corrente, e nella parte superiore o inferiore della finestra di visualizzazione.

Scorrimento Lo spostamento del testo verso l'alto o verso il basso necessario per rendere visibili delle porzioni di testo precedentemente nascoste, è chiamato scorrimento. Ci sono dei comandi che consentono di far scorrere il programma verso l'alto o verso il basso di una riga alla volta, altri di un'intera schermata, e altri ancora all'inizio e alla fine del programma.

Ricerca e sostituzione Con l'editor si possono ricercare determinate occorrenze di testo all'interno del programma, e si possono anche sostituire le stringhe localizzate con un altro testo specificato (sia automaticamente che manualmente).

Gestione dei blocchi Si possono contrassegnare dei blocchi di testo all'interno del programma e spostare o copiare il blocco selezionato in un'altra posizione. Quando si contrassegna un blocco, questo viene evidenziato. È inoltre possibile cancellare un blocco o stamparlo.

Cancellazione I tasti Backspace e Canc consentono di cancellare un carattere alla volta. L'editor dispone di comandi che permettono di cancellare la riga su cui è posizionato il cursore, tutto il testo che si trova tra la posizione del cursore e la fine della riga, o la parola a destra del cursore.

Recupero Se una porzione di testo viene cancellata involontariamente, è possibile recuperarla. In questo modo non si corre il rischio di perdere del lavoro quando, dopo aver selezionato un blocco di testo, si preme inavvertitamente il tasto sbagliato.

Salvare e caricare i programmi Si può salvare su disco un qualsiasi programma scritto con l'editor e caricarlo successivamente in memoria.

LA TASTIERA

Dato che è stato ripetuto più volte che i programmi vengono inseriti da tastiera, si ne esaminerà ora il funzionamento.

Ci sono alcuni tipi differenti di tastiera. La Figura 3.1 mostra una tipica tastiera di un PC IBM compatibile. La tastiera è divisa in tre parti. La porzione centrale equivale ad una normale tastiera di una macchina da scrivere.

La porzione a sinistra consiste di 10 tasti identificati da F1 a F10 denominati *tasti funzione*. In alcune tastiere questi tasti sono situati nella parte superiore e ci sono più di dieci tasti funzione. I tasti funzione vengono utilizzati per eseguire determinate operazioni con la singola pressione di un tasto. Per esempio, la pressione del tasto F1 attiva il sistema di aiuto. In questa sezione si esamineranno alcuni dei comandi associati ai tasti funzione.

Il gruppo di tasti situato nella parte di destra della tastiera è conosciuto come *tastierino numerico* e viene utilizzato per spostare il cursore o per inserire dei numeri. Si prema alcune volte il tasto Bloc Num e si noti come la lettera N viene visualizzata e rimossa dalla barra di stato nella parte inferiore dello schermo. Quando la lettera N è visualizzata, i tasti del tastierino generano dei numeri; in caso contrario, spostano il cursore. Il tasto Bloc Num funge da *interruttore* nel senso che attiva uno dei due stati possibili. Quando il tastierino si trova in modalità di spostamento, i quattro tasti freccia spostano il cursore di uno spazio nella direzione corrispondente.

Ci sono due tasti molto importanti che potrebbero non avere un nome stampato sul tasto stesso. Il tasto Invio, che viene utilizzato per eseguire dei comandi o inserire delle righe di programma, può essere identificato con una freccia ad angolo retto, mentre il tasto Backspace, situato sopra al tasto Invio e utilizzato per cancellare i caratteri che si trovano a sinistra del cursore, è identificato da una freccia a sinistra.

Se si desidera fare un po' di esercizio con la tastiera, si avvii QBasic e si proceda nel modo seguente:

1. usare i tasti Destra e Sinistra sul tastierino numerico per spostare il cursore. Si noti che il numero che identifica la posizione del cursore nell'angolo in basso a destra dello schermo cambia ogni volta che si preme uno di questi tasti;
2. premere il tasto Home per spostare il cursore all'inizio della riga. In generale, il tasto Home sposta il cursore all'estrema sinistra della riga sui cui questo è posizionato;
3. digitare alcune lettere utilizzando i tasti situati nella porzione centrale della tastiera. I tasti Maiusc, identificati con una freccia rivolta verso l'alto, consentono di inserire le lettere in maiuscolo, o il simbolo riportato nella parte superiore di un tasto che mostra due caratteri;

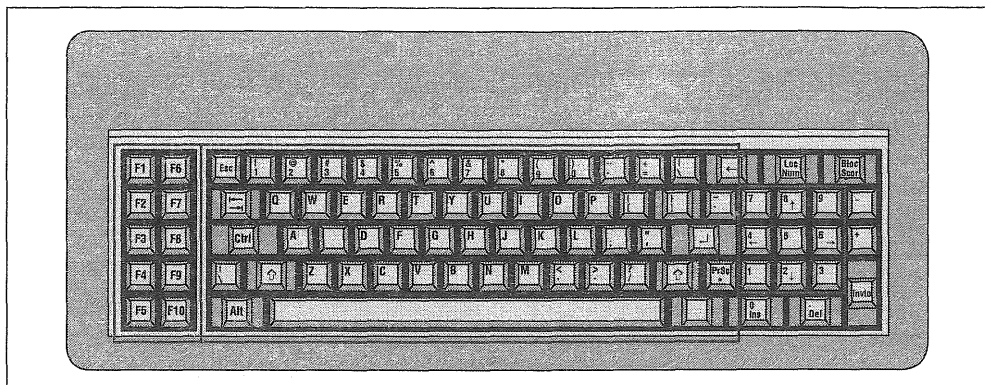



Figura 3.1 *La tastiera del PC IBM*

4. premere il tasto Bloc Maiu e digitare alcune lettere; queste appaiono in maiuscolo. Anche questo tasto funge da interruttore, e consente di passare tra la *modalità di inserimento maiuscole* e quella standard. Si noti, tuttavia, che il tasto Bloc Maiu agisce solo sui tasti alfabetici. Quando l'editor si trova in modalità di inserimento maiuscole, la lettera M viene visualizzata nella parte di destra della barra di stato;
5. tenere premuto il tasto Ctrl (Ctrl sta per Control) e premere il tasto Y. Questa combinazione di tasti cancella la riga su cui è posizionato il cursore e viene normalmente identificata come Ctrl-Y;
6. digitare alcune lettere e premere quindi Backspace alcune volte. Ogni volta che si preme questo tasto, viene cancellato il carattere che si trova a sinistra del cursore. Un altro metodo per cancellare un carattere, è quello di spostare il cursore sulla lettera da rimuovere e premere il tasto Canc. Esiste una differenza tra questi due metodi: Backspace cancella il carattere a sinistra del cursore, mentre Canc elimina il carattere che si trova alla posizione del cursore;
7. digitare alcune lettere ed utilizzare i tasti freccia appropriati per spostare il cursore sotto una di queste lettere. Digitare ora un altro carattere; si noti che questo viene inserito nel punto in cui è posizionato il cursore e che le lettere che seguono si spostano verso destra. Ciò accade perché è attiva la modalità di *inserimento*. Se si preme il tasto Ins, viene attivata la modalità di *sovrascrittura* in cui i caratteri digitati sovrascrivono quelli esistenti. Premendo nuovamente il tasto Ins, viene riattivata la modalità di inserimento. Si può determinare la modalità correntemente attiva dalla dimensione del cursore: un cursore più largo indica che ci si trova in modalità di sovrascrittura;
8. digitare delle altre lettere e spostare il cursore a sinistra di alcuni spazi. Si prema ora il tasto Fine. Il cursore si sposta alla fine della riga;
9. il tasto che si trova a sinistra della lettera Q è chiamato tasto Tab, ed è identificato da una coppia di frecce; la freccia nella parte superiore del tasto

punta verso sinistra, mentre quella nella parte inferiore verso destra. Nella finestra di visualizzazione, la pressione del tasto Tab ha lo stesso effetto di una serie di pressioni del tasto spazio;

10. digitare più caratteri di quanti possano essere contenuti in una sola riga dello schermo. Si noti che i caratteri più a sinistra scorrono fuori dallo schermo per accomodare i nuovi caratteri digitati. Una riga può contenere un massimo di 255 caratteri. Alcuni dei programmi riportati in questo libro contengono delle righe di istruzioni molto lunghe. Dato che non è stato possibile stamparle sulla pagina come righe uniche, sono state divise in due porzioni ed è stato utilizzato il simbolo di sottolineatura (_) per indicare di digitarle come unica riga;
11. il tasto Invio viene utilizzato per iniziare una nuova riga ed equivale alla levetta di ritorno a capo di una macchina da scrivere. La pressione del tasto Invio, tuttavia, sottopone la riga in questione ad un'ulteriore analisi. QBasic, infatti, esamina il contenuto della riga e mostra un messaggio nel caso rilevi un errore. Se le istruzioni sono corrette, converte automaticamente tutte le parole chiave in lettere maiuscole;
12. il tasto Alt attiva la barra dei menu da cui è possibile selezionare la voce desiderata premendo la lettera evidenziata corrispondente. Un menu può anche essere selezionato utilizzando il tasto Destra, per evidenziarne il nome, e premendo quindi Invio. Come mostrato nella Figura 3.2, tutti i comandi contenuti in un menu hanno una lettera evidenziata che è quella da premere per impartire quel determinato comando. Per esempio, se si preme la lettera V dopo aver aperto il menu File, si impartisce il comando Salva con nome. Anche in questo caso, tuttavia, le selezioni possono essere effettuate tramite i tasti cursore e il tasto Invio;
13. il tasto Esc viene utilizzato per ritornare alla finestra di visualizzazione dalla barra dei menu senza selezionare alcun comando.

COMANDI DI REVISIONE

Il motivo per cui si è detto che gli utenti in grado di utilizzare WordStar non dovrebbero incontrare dei problemi nell'utilizzo dell'editor di QBasic, dipende dal fatto che i comandi di revisione di QBasic sono stati pesantemente influenzati da questo programma di elaborazione testi. Quasi ogni comando può essere impartito premendo il tasto Ctrl in combinazione con un altro tasto. Per esempio, la combinazione di tasti Ctrl-F sposta il cursore sul primo carattere della parola successiva a destra, mentre la combinazione di tasti Ctrl-Q seguita dalla pressione del tasto D sposta il cursore alla fine della riga che lo contiene (questa combinazione viene espressa come Ctrl-Q/D). Il tasto Ctrl deve rimanere premuto solo quando si preme il tasto Q. Entrambi i tasti possono essere rilasciati quando si preme la lettera D.

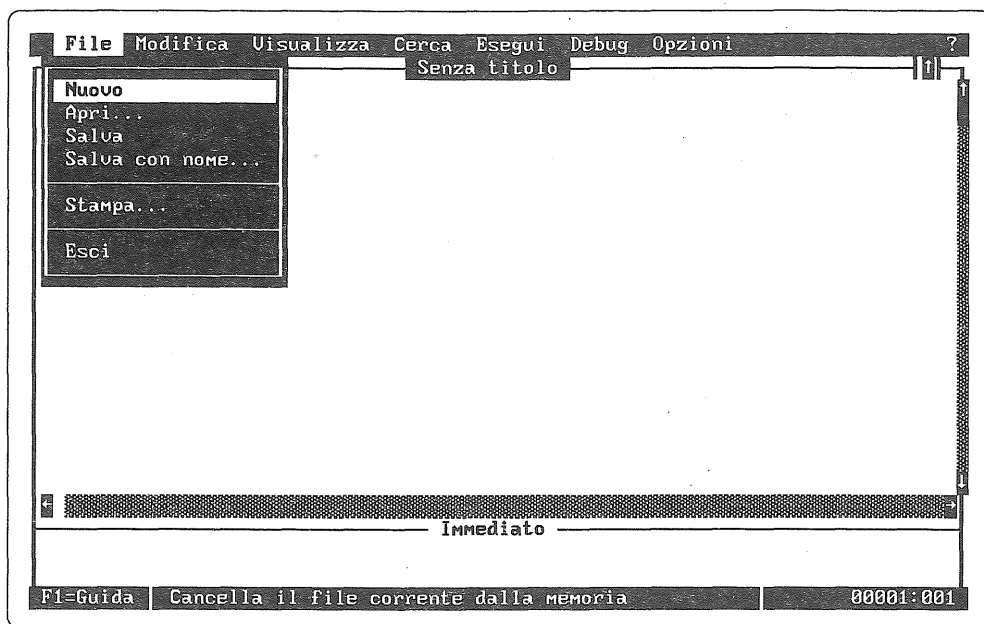



Figura 3.2 *Un menu e le sue opzioni*

La Tabella 3.1 riporta alcune delle combinazioni di tasti più comunemente usate nell'editor.

USO DI APPUNTI

Ci si potrebbe chiedere che cosa accade quando si cancella del testo. QBasic non elimina effettivamente il testo, ma lo rimuove semplicemente dal programma e lo conserva in un'area di memoria temporanea denominata Appunti. Quest'area di memoria viene anche utilizzata per gestire tutte le operazioni di *copia* e *spostamento* del testo.

Nel momento in cui si preme la combinazione di tasti Ctrl-Y, il testo cancellato viene inserito in Appunti. A questo punto, in qualsiasi momento è possibile premere i tasti Maiusc-Ins per *inserire* il contenuto di Appunti alla posizione del cursore. Quindi, si può cancellare (tagliare) del testo in una posizione e inserirlo (incollarlo) in un'altra.

Per esempio, si proceda nel modo seguente per spostare una riga di istruzioni in una nuova posizione:

1. posizionare il cursore in un punto qualsiasi della riga da spostare;

Tabella 3.1 *I comandi di revisione più comuni*

Per spostare il cursore	**Premere**
a sinistra di un carattere	Sinistra
a destra di un carattere	Destra
all'inizio della parola a sinistra	Ctrl-Sinistra o Ctrl-A
all'inizio della parola a destra	Ctrl-Destra o Ctrl-F
all'inizio della riga	Home o Ctrl-Q/S
alla fine della riga	End o Ctrl-Q/D
in alto di una riga	Alto
in basso di una riga	Basso
sulla prima riga del programma	Ctrl-Home
sull'ultima riga del programma	Ctrl-End
Per scorrere	**Premere**
in alto di una pagina	PgUp
in basso di una pagina	PgDn
in alto di una riga	Ctrl-Alto
in basso di una riga	Ctrl-Basso
Per cancellare	**Premere**
il carattere a sinistra del cursore	Backspace
il carattere alla posizione del cursore	Canc
il testo tra il cursore e la fine della parola	Ctrl-T
il testo tra il cursore e la fine della riga	Ctrl-Q/Y
la riga su cui è posizionato il cursore	Ctrl-Y

2. premere Ctrl-Y per cancellare l'intera riga e copiarla in Appunti;
3. spostare il cursore nella nuova posizione desiderata;
4. premere Maiusc-Ins per inserire la riga di istruzioni alla posizione del cursore.

Nota: Dopo aver copiato il contenuto di Appunti, il testo *resta in Appunti* e può essere duplicato in altre posizioni.

Si possono copiare in Appunti anche porzioni contigue di testo, denominate *blocchi*. Per selezionare una parte di testo come blocco, si sposti il cursore all'inizio del testo

da selezionare, si tenga premuto il tasto Maiusc, si sposti il cursore alla fine del blocco, e si rilasci il tasto Maiusc. Il blocco selezionato viene evidenziato. I comandi seguenti agiscono sul blocco di testo selezionato:

Maiusc-Canc Rimuove il blocco selezionato dalla posizione corrente e lo copia in Appunti.

Ctrl-Ins Copia il blocco selezionato in Appunti lasciando l'originale intatto.

Canc Cancella il blocco selezionato senza inserirlo in Appunti.

Se si esegue una qualsiasi operazione diversa da una delle tre sopra citate, si sposta ad esempio il cursore, la porzione evidenziata viene deselezionata. Ciò significa che il blocco non è più evidenziato e che non si possono più svolgere delle operazioni su quel blocco specifico. Per spostare un blocco di testo:

1. selezionare il blocco da spostare usando il tasto Maiusc e i tasti cursore;
2. premere Maiusc-Canc per tagliare il blocco e inserirlo in Appunti;
3. spostare il cursore nella posizione in cui si desidera inserire il blocco;
4. premere Maiusc-Ins per inserire il blocco alla posizione del cursore.

Questo capitolo dovrebbe aver dato un'idea dei comandi disponibili nell'editor e di come funzioni l'editor stesso. Tuttavia, prima di passare alla programmazione vera e propria, si dovrebbe seguire l'esercizio sotto riportato per mettere in pratica queste nuove nozioni.

UN ESERCIZIO DI EDITING

L'esercizio seguente introduce i comandi usati più di frequente. Si inizi con un nuovo programma, usando Ctrl-Y oppure Alt/F/N per cancellare il contenuto della finestra di visualizzazione, e si digiti la riga seguente includendo gli errori appositamente inseriti:

```
PRIMT "La ata è"; DATE$
```

Il cursore dovrebbe ora trovarsi a destra del simbolo del dollaro. Si prema Home, oppure Ctrl-Q/S. Si noti che il cursore si sposta all'inizio della riga ed è posizionato sulla lettera P della parola PRIMT.

Si cambi la lettera M in N, in modo che il comando diventi PRINT. A questo scopo, si prema il tasto Destra tre volte per spostare il cursore sulla lettera M, si cancelli la M premendo Canc e si digiti N per completare la correzione.

Si aggiunga quindi uno spazio dopo la parola "è". Si prema Ctrl-Destra o Ctrl-F quattro volte. Si noti che il cursore si sposta all'inizio della parola a destra ogni volta che si preme questa combinazione di tasti. Il cursore dovrebbe trovarsi ora sulla lettera D della variabile DATE$. Si prema il tasto Sinistra tre volte per posizionare il cursore sulle virgolette e si prema la barra spaziatrice per inserire uno spazio dopo la parola "è".

A questo punto, si corregga la parola "ata" in "data". Si prema due volte Ctrl-Sinistra o Ctrl-A; il cursore si sposta all'inizio della parola a sinistra ogni volta che si preme questa combinazione di tasti. Il cursore dovrebbe ora trovarsi sulla lettera "a" della parola "ata". Si digiti d per comporre la parola data.

Si prema End oppure Ctrl-Q/D per spostare il cursore alla fine della riga e si prema Invio per creare una nuova riga (si faccia attenzione a non premere Invio se il cursore non è posizionato alla fine della riga). Si digiti ora il comando seguente:

```
PRINT "Il tempo non aspetta nessuno."
```

Si sposti il cursore sulla prima riga premendo il tasto Alto. Si noti che il cursore rimane nella stessa colonna. Si riporti il cursore sulla seconda riga premendo Basso, e si prema Ctrl-Sinistra quattro volte per spostare il cursore sulla parola "tempo". Si prema Ctrl-T; l'editor cancella la parola tempo. Si prema nuovamente Ctrl-T per cancellare lo spazio che separava la parola "tempo" dalla parola "non", e ancora Ctrl-T per rimuovere la parola non. La combinazione di tasti Ctrl-T cancella il testo presente tra la posizione del cursore e la fine della parola o del gruppo di spazi corrente.

Si supponga di aver cancellato inavvertitamente le parole "tempo non"; invece di ridigitarle, si può utilizzare la funzione di recupero disponibile nell'editor di QBasic. A questo scopo, si prema la combinazione di tasti Ctrl-Q/L; la riga verrà riportata nello stesso stato in cui si trovava prima della cancellazione del testo.

A questo punto, si sposti il cursore all'inizio della parola "non" e si cancelli la parte rimanente della riga premendo la combinazione di tasti Ctrl-Q/Y. Si digiti "è denaro" aggiungendo le virgolette di chiusura per completare la modifica della riga. Il cursore dovrebbe ora trovarsi alla fine della riga seguente

```
PRINT "Il tempo è denaro"
```

Si combineranno ora le funzioni di gestione dei blocchi di testo con il sistema di aiuto in linea. Le finestre di aiuto generate da QBasic contengono spesso degli esempi di programmi. Questi programmi possono essere tagliati dalla finestra di aiuto, incollati nella finestra di visualizzazione ed utilizzati nei programmi di QBasic.

A titolo di esempio, si cancelli la finestra di visualizzazione, si digiti la parola FOR senza premere Invio, e si prema F1. Apparirà una finestra di aiuto con delle informazioni di aiuto relative alle istruzioni FOR...NEXT. Si noti che la finestra di visualizzazione rimane sullo schermo, ma viene ridotta ad una singola riga. La

schermata di aiuto include un esempio nella parte inferiore della finestra; si utilizzerà questo esempio come codice di programma.

Si prema F6 due volte per attivare la finestra di aiuto, e si utilizzi quindi il tasto Basso per spostare il cursore sulla riga che si trova sotto alla parola Esempio. Questa riga inizia con il testo che si vuole copiare. Si tenga premuto il tasto Maiusc e si sposti il cursore verso il basso fino ad evidenziare le tre righe di programma. Si rilasci il tasto Maiusc e si impartisca il comando Alt/M/O, per selezionare Copia dal menu Modifica. Il testo evidenziato viene copiato in Appunti.

Si prema Esc per ritornare alla finestra di visualizzazione e quindi Alt/F/N per selezionare Nuovo dal menu File. Si risponda No quando QBasic richiede se si desiderano salvare le modifiche apportate al programma corrente. A questo scopo, si prema Tab per selezionare No e si prema Invio. Nella nuova finestra di visualizzazione, digitare innanzi tutto CLS e premere Invio. Con questo comando si indica al programma di cancellare il contenuto dello schermo prima dell'esecuzione. Si prema quindi Alt/M/N per selezionare Incolla dal menu Modifica ed incollare il contenuto di Appunti nella finestra di visualizzazione. Si prema nuovamente Alt/F/N per inserire una seconda copia e quindi Maiusc-F5 per eseguire il programma. Questo programma conta per due volte da 1 a 15.

Si supponga di volere che il programma conti fino a 15 una sola volta. Si prema un tasto qualsiasi per ritornare alla finestra di visualizzazione e si sposti il cursore all'inizio del secondo enunciato FOR. Si selezionino le ultime tre righe del programma tenendo premuto il tasto Maiusc e premendo tre volte Basso, e si cancelli il blocco impartendo il comando Cancella del menu Modifica.

L'unico comando relativo ai blocchi che non è stato provato è il comando Taglia, che agisce come Cancella ad eccezione del fatto che copia il testo cancellato dal programma in Appunti. Questa opzione viene usata per spostare un blocco di testo.

Come ultima operazione di editing si utilizzerà il comando Cambia. Si prema Alt/C/C per impartire il comando Cambia del menu Cerca. Con questo comando si può ricercare una determinata stringa e sostituirla con dell'altro testo specificato.

Si cambierà ora la variabile i% che non risulta molto descrittiva. Dopo aver impartito il comando cambia, appare una finestra di dialogo in cui si possono specificare i parametri per l'operazione di ricerca e sostituzione. Si digiti i% nella casella Trova, si prema il tasto Tab, si digiti conta% nella casella Cambia in e si prema Invio. L'editor evidenzia la prima occorrenza della stringa i%; si prema Invio per eseguire la sostituzione e si ripeta questa procedura altre due volte, in modo da cambiare tutte le occorrenze di i%. Si noti che si sarebbero potute effettuare le tre sostituzioni con un solo comando selezionando Cambia tutto dalla finestra di dialogo iniziale.

Si acquisirà una certa familiarità con le funzioni dell'editor di QBasic dopo aver scritto qualche programma, e molto probabilmente si finirà col memorizzare le diverse

scorciatoie associate ai comandi di menu. Gli argomenti trattati in questo capitolo sono sufficienti per iniziare a lavorare; tuttavia, l'editor dispone di molti altri comandi che mettono a disposizione funzioni decisamente interessanti. Si faccia riferimento all'Appendice E per una spiegazione sugli altri comandi dell'editor.

A questo punto si può finalmente esaminare ciò che QBasic è in grado di offrire. Come detto in precedenza, tutti i linguaggi di programmazione consentono di gestire dei dati, ed è quindi ovvio che un computer deve disporre di un'area in cui memorizzare i dati e i risultati.

Prima di addentrarsi nei comandi specifici di QBasic, quindi, si esamineranno i modi in cui QBasic memorizza i dati. Si introdurrà il concetto di *variabile,* che sta alla base della programmazione, e si prenderanno in considerazione i diversi tipi di variabili disponibili in QBasic. Una volta apprese queste nozioni, si potrà iniziare a sfruttare pienamente le potenzialità offerte da questo linguaggio di programmazione.

CAPITOLO 4

GESTIONE DEI DATI

INTRODUZIONE

Si è visto come utilizzare l'editor di QBasic per creare e modificare dei programmi. Tuttavia, sono state esaminate solamente le funzioni di base, e sono stati appena accennati alcuni dei comandi di questo linguaggio. Inoltre, non sono state ancora approfondite le potenzialità e gli strumenti messi a disposizione da QBasic.

Prima di procedere con il linguaggio di programmazione vero e proprio, si dovrà esaminare la 'materia' su cui opera un programma. Come in una carpenteria viene trattato il legno, così in un computer vengono elaborati i dati e, come il legno, anche i dati si presentano in forme differenti. Anche all'interno del computer, i dati possono essere memorizzati in modi differenti. Tutto ciò si verifica sotto il controllo dell'utente. Rilevare il tipo di memorizzazione appropriato per una determinata serie di dati è un compito importante per il programmatore. In questo capitolo si prenderanno in esame i diversi metodi disponibili per la memorizzazione dei dati, e si vedrà come gestirli attraverso calcoli aritmetici e operazioni su stringhe di testo. Infine, si imparerà ad inserire i dati nel computer, e a prelevarli quando necessario.

I DATI DEL COMPUTER

Si pensi a tutte le operazioni per cui si potrebbe programmare un computer. Si potrebbe elaborare del testo per formattare in modo preciso un documento, si potrebbero ordinare lunghi elenchi di nomi e indirizzi, e si potrebbero eseguire calcoli matematici complessi.

Tutte queste operazioni possono essere eseguite sugli stessi dati; dopo tutto, il computer 'vede' semplicemente delle lunghe stringhe composte da zero e uno. Teoricamente, si potrebbe sommare un elenco di nomi e ordinare una lista di valori numerici; ciò è possibile anche se non desiderabile. Per questi motivi, bisogna trovare un modo per indicare al computer quale tipo di dati si intende trattare, e se una determinata stringa deve essere considerata come testo o come valore.

Nella memoria del computer, i dati vengono conservati in locazioni predefinite. È tuttavia molto importante ricordare che è il programmatore che definisce queste locazioni. Una buona parte di questo capitolo mostra come svolgere questa operazione. Per ora, è sufficiente menzionare che ci sono due categorie di locazioni per la memorizzazione dei dati: le costanti e le variabili.

Le locazioni costanti contengono semplicemente un valore che non cambia. Come esempio, si consideri il valore di pi greco troncato al quinto decimale: 3,14159. Quando si inserisce una costante in un programma, il computer riserva uno spazio fisso nella memoria e lo riempie con la sequenza binaria che corrisponde al numero specificato. Tutto ciò che bisogna fare è inserire il numero desiderato nel punto appropriato. Per esempio, se si volesse moltiplicare pi per due, si dovrebbe scrivere:

```
3,14159*2
```

Questa espressione non è molto interessante (un'espressione è un problema in cui vengono utilizzati simboli matematici); sarebbe molto più utile se consentisse di moltiplicare pi per un valore specificato dall'utente, senza dover ogni volta cambiare la riga. Inoltre, è necessaria una locazione per memorizzare il risultato. Ciò introduce il concetto di *variabile.*

Una variabile è una locazione di memoria predefinita che può contenere un qualsiasi valore. Inoltre, un valore contenuto in una variabile può essere modificato in qualsiasi momento. Ad ogni variabile in un programma viene assegnato un nome. Questo nome può essere paragonato ad un'etichetta che viene incollata su un barattolo per indicarne il contenuto. Si consideri una forma più generale dell'espressione presentata in precedenza: la formula per calcolare la circonferenza di un cerchio, dato il raggio, è:

*C=2*pi*r*

C indica la circonferenza, che è l'incognita da calcolare, 2 è una costante, pi è un'altra costante uguale, approssimativamente, a 3,14159, e r è il raggio del cerchio, cioè la distanza dal centro del cerchio al suo bordo. Questo valore è conosciuto, anche se cambia a seconda del cerchio.

A questo punto, se si volesse scrivere un programma per calcolare la circonferenza di un cerchio, si dovrebbero impostare due variabili: una per contenere il risultato (la circonferenza), e l'altra per contenere il raggio del cerchio su cui effettuare l'operazione. In questo modo, si potrebbe calcolare la circonferenza di un cerchio qualsiasi.

Con le versioni precedenti del BASIC, per creare una variabile è sufficiente specificarla in un'espressione. Quando l'interprete incontra per la prima volta il nome di una variabile, la crea automaticamente. Anche in QBasic si può procedere in questo modo, benché non si tratti di una buona regola di programmazione. Ci sono infatti diversi tipi di variabili, tra cui quelle che contengono numeri *interi*, come 2500 e -5, quelle che contengono frazioni, e quelle che contengono del testo.

Se non viene specificato il tipo di variabile che si intende utilizzare, QBasic utilizza quello che meglio si adatta al valore che si sta attualmente inserendo. Tuttavia, questa supposizione potrebbe essere sbagliata perché QBasic non può 'sapere' quali dati si memorizzeranno successivamente in quella variabile. Ad esempio, se QBasic utilizzasse una variabile intera per memorizzare il valore 3, e successivamente si volesse inserire nella stessa variabile il valore 3,14159, la parte decimale di questo numero andrebbe persa, dato che una variabile intera accetta solo numeri interi. I tipi di variabile verranno spiegati più avanti in questo capitolo.

È ovviamente possibile inserire e prelevare dei dati da una qualsiasi variabile. Tutti i linguaggi di programmazione mettono a disposizione delle istruzioni per queste operazioni che ricadono nella categoria generale di operazioni di *Input/Output* (spesso abbreviato in I/O).

Le istruzioni più comunemente utilizzate per inserire un valore in una variabile sono LET e INPUT. L'enunciato

```
LET variabile = valore
```

assegna il valore specificato alla variabile. La parola chiave LET può essere omessa. Altri esempi di assegnazioni di variabile sono:

```
peso = 4,25

totale = 4 + 5
```

L'enunciato nella forma

```
INPUT "richiesta", variabile
```

visualizza il messaggio *richiesta*, attende che l'utente digiti una risposta e prema il tasto Invio, ed inserisce la risposta nella variabile. Segue un esempio di questa istruzione:

```
INPUT "Quanti anni hai? ", età
```

PRINT è l'istruzione di output maggiormente utilizzata. L'enunciato

```
PRINT valore
```

visualizza il valore specificato sullo schermo, mentre l'enunciato

```
PRINT "testo"
```

mostra la stringa di testo a video. Per esempio, i due enunciati seguenti

```
PRINT "La somma è"
PRINT 2+3
```

visualizzano rispettivamente

```
La somma è
 5
```

L'istruzione PRINT visualizza i numeri positivi con uno spazio in testa, ed aggiunge uno spazio in coda a tutti i numeri.

Quando l'istruzione PRINT è seguita da una serie di elementi separati da un punto e virgola, questi vengono visualizzati sulla stessa riga, uno dopo l'altro. Ad esempio, l'enunciato

```
PRINT "La somma è"; 2+3
```

genera l'output seguente

```
La somma è 5
```

Si dovrebbe ora avere un'idea generale di come i dati vengono richiesti, memorizzati e visualizzati. Si tratteranno ora argomenti specifici, iniziando dai dati numerici e dalle operazioni che si possono svolgere con essi.

NUMERI

QBasic è molto flessibile nella gestione dei numeri e consente di effettuare calcoli con numeri compresi tra 10^{-324} e 10^{308}. Si noti che l'ultimo numero equivale a 10 seguito da

307 zeri. Per rendersi pienamente conto di questa grandezza, si pensi che nell'universo conosciuto ci sono particelle nell'ordine di 10^{100}. Se si allineassero tutte queste particelle e si costruisse un cubo con un lato della stessa lunghezza di quella linea di particelle, il cubo potrebbe contenere 10^{300} particelle. 10^{308} è più grande di 100 milioni di volte di quel numero.

Come detto in precedenza, un numero può essere utilizzato in QBasic sia direttamente, come valore numerico (ad esempio 3,14159), che indirettamente, come variabile contenente un numero. Ci si ricordi che una variabile è una locazione di memoria a cui viene assegnato un nome che può essere utilizzato negli enunciati del programma.

Si parlerà dei quattro tipi di variabili numeriche nella prossima sezione. Al momento, è sufficiente sapere che tutti i nomi di variabile devono iniziare con una lettera e che possono essere composti da lettere, numeri e punti decimali fino a un massimo di 40 caratteri. Le lettere maiuscole e minuscole vengono considerate allo stesso modo; ad esempio, pi e PI fanno riferimento alla stessa variabile.

Ci sono alcune parole che non possono essere utilizzate come nomi di variabile. Ci si ricorderà che si è parlato di *parole riservate* che hanno un significato speciale in QBasic; queste non possono essere utilizzate. L'Appendice F riporta l'elenco delle parole riservate che non possono essere utilizzate come nomi di variabile.

Tabella 4.1 *Operatori aritmetici*

Operatore	Nome	Esempio
+	Addizione	2 + 3 uguale 5
-	Sottrazione	3 - 2 uguale 1
*	Moltiplicazione	2 * 3 uguale 6
/	Divisione	2 / 3 uguale 0,6666667
^	Esponenziale	3^2 uguale 9
-	Negativo	-(1,5) uguale -1,5
\	Divisione intera	17\5 uguale 3
MOD	Modulo	17 MOD 5 uguale 2

Un buon nome di variabile deve essere descrittivo ed indicare come viene utilizzata la variabile e quale tipo di dati contiene. Alcuni esempi di nomi di variabile sono costoDelMateriale, chilometriPercorsi, risposta e nomeDelGiocatore. In questo libro, le variabili vengono scritte in lettere minuscole. Se un nome di variabile è composto da più di una parola, come chilometriPercorsi, l'iniziale di ogni nuova parola viene presentata in maiuscolo; ciò viene fatto solo per facilitare la lettura.

Per gestire dei dati numerici, indipendentemente dal fatto che si trovino sotto forma di costanti o di variabili, QBasic mette a disposizione gli otto operatori aritmetici

riportati nella Tabella 4.1. Nella prossima sezione ci si soffermerà sugli operatori *divisione intera* e *modulo* che sono meno conosciuti degli altri.

DIVISIONE INTERA E MODULO

Quando si dividono due numeri interi tra loro non multipli, si ottiene un quoziente intero e un resto. Per esempio, 122 diviso 5 genera il quoziente intero 24 e il resto 2.

```
       24
5  | 122
     10
     ---
      22
      20
     ---
       2
```

In generale, se *M* e *N* sono due numeri interi tra loro non multipli,

```
N\M
```

denota il quoziente intero, e

```
N MOD M
```

denota il resto. Ad esempio, 122\5 è uguale a 24, e 122 MOD 5 è uguale a 2.

QBasic esegue la divisione intera e l'operazione MOD su qualsiasi coppia di numeri compresi tra -2.147.483.648 e +2.147.483.647. Se *M* o *N* non sono dei numeri interi, QBasic li arrotonda all'intero più vicino. Il valore di *N**M* equivale alla porzione intera del quoziente che deriva dall'operazione *N/M*, mentre il valore di *N* MOD *M* equivale a *N* - *M* * (*N**M*). I numeri *M* e *N* possono essere sostituiti con variabili numeriche di qualsiasi tipo. Se i numeri utilizzati in queste operazioni non sono compresi nell'intervallo di valori appropriato, viene generato il *messaggio di errore* **Overflow**. Un messaggio di errore appare ogni volta che si verifica un problema ed indica quale tipo di errore è stato rilevato.
Il Programma 4.1 utilizza gli operatori \ e MOD per calcolare la quantità di nickel e penny necessaria per formare il totale specificato dall'utente.

Nota: Il messaggio di errore **Overflow** viene visualizzato quando si esegue il Programma 4.1 specificando il valore 3000000000.

Nei listati riportati in questo libro, si utilizzerà la dicitura [Esecuzione] per indicare la pressione della combinazione di tasti Maiusc-F5.

Programma 4.1 *Divisione intera e MOD*

```
REM Divisione intera e MOD [4-1]
INPUT "Quantità in centesimi: "; money
nickels = money \ 5
pennies = money MOD 5
PRINT nickels; "nickel e"; pennies; "penny"
END

[Esecuzione]
Quantità in centesimi: 23
 4 nickel e 3 penny

[Esecuzione]
Quantità in centesimi: 3000000000
{ Overflow }
```

Il Programma 4.1 inizia con l'enunciato REM e termina con l'istruzione END. REM sta per remark (annotazione). Gli enunciati REM non vengono eseguiti e servono solamente per inserire dei commenti e delle spiegazioni. L'istruzione END indica la fine del programma. I dati che devono essere forniti dall'utente in risposta al comando INPUT sono riportati in corsivo.

TIPI DI VARIABILI

Sono state esaminate alcune delle operazioni per la gestione dei numeri disponibili in QBasic. Si prenderanno ora in esame i tipi di variabili numeriche forniti da QBasic.

Per ottimizzare il tempo di esecuzione e l'uso della memoria, QBasic dispone di quattro tipi differenti di variabili numeriche. Questi tipi sono conosciuti come *intero, intero lungo, virgola mobile a precisione singola,* e *virgola mobile a precisione doppia.* Il tipo di una variabile numerica deve essere specificato con un simbolo di dichiarazione (%, &, !, #) posto alla fine del nome della variabile. Se non viene specificato un simbolo di dichiarazione, QBasic crea una variabile a precisione singola. Questa impostazione predefinita viene chiamata default. Si dice quindi che il tipo di variabile di default è a precisione singola.

VARIABILI INTERE E INTERE LUNGHE

Le variabili intere e intere lunghe possono contenere solamente numeri interi, sia positivi che negativi, compresi in un determinato intervallo. Una variabile intera può contenere numeri interi compresi tra -32.768 e 32.767, mentre una variabile intera lunga è in grado di memorizzare numeri compresi tra -2.147.483.648 e 2.147.483.647.

La ragione è semplice. Una variabile intera occupa due byte nella memoria del computer, mentre una variabile intera lunga ne occupa quattro.

Programma 4.2 *Variabili intere e intere lunghe*

```
REM Variabili intere e intere lunghe [4-2]
castOfBenHur% = 25452
wordsInBible& = 773692
US1990MedianAge% = 31.6
PRINT US1990MedianAge%
US1990NationalDebt& = 3000000000000
END

[Esecuzione]
 32
{ Overflow }
```

Per definire una variabile come intera o intera lunga, bisogna aggiungere, rispettivamente, il simbolo di dichiarazione % o &. Benché gli interi lunghi richiedano più memoria e più tempo per essere elaborati, risultano spesso necessari. Per esempio, il numero di persone che vivono a Milano non potrebbe essere contenuto in una variabile intera. Quando si assegna ad una variabile intera, o intera lunga, un numero che supera il limite corrispondente, viene generato un errore di overflow, analogo a quello visto nel Programma 4.1.
Quando un numero *non* intero (come 2,5 o 3,14159) viene assegnato ad una variabile intera o intera lunga, il numero viene arrotondato all'intero più vicino. Se la parte decimale del numero è uguale a 5, l'arrotondamento viene effettuato sulla base del numero pari più vicino. Ad esempio, 2,5 viene arrotondato a 2, mentre 3,5 a 4.

Il Programma 4.2 assegna dei valori a delle variabili numeriche e visualizza il valore di una delle variabili. L'ultimo enunciato genera il messaggio di errore Overflow perché cerca di assegnare a una variabile intera lunga un numero troppo grosso.

VARIABILI A VIRGOLA MOBILE

Le variabili intere sono ideali per memorizzare valori che non superino i 2 miliardi. Ci sono tuttavia diverse situazioni in cui risulta necessario gestire numeri ben superiori a questa cifra. Per memorizzare numeri molto grossi, e valori frazionari, sono necessari altri due tipi di variabile. QBasic dispone di due tipi di variabile a virgola mobile che differiscono tra loro per la precisione e la grandezza dei valori che possono gestire. Come accade per i due tipi di variabili intere, le variabili a precisione doppia occupano più memoria di quelle a precisione singola.

Nota: I numeri espressi in notazioni scientifica sono esempi di numeri in virgola mobile (per virgola si intende la virgola decimale). Un numero in virgola mobile è un

numero scritto nella forma $m*10^e$, dove m rappresenta la *mantissa* ed e l'*esponente*. La mantissa contiene normalmente una virgola decimale all'estrema sinistra, o dopo una cifra da sinistra, mentre l'esponente è sempre un numero intero positivo, negativo, o uguale a zero. Il numero di *cifre significative* nel numero in virgola mobile corrisponde al numero di cifre da cui è composta la mantissa. Per esempio, 186.000 può essere espresso in virgola mobile come $1{,}86*10^5$, dove 1,86 rappresenta la mantissa e 5 l'esponente; in questo caso ci sono tre cifre significative. Come ulteriore esempio, si consideri il numero $1{,}2*10^{-15}$ che equivale al numero 0,0000000000000012. La notazione in virgola mobile consente di rappresentare dei numeri molto grossi o molto piccoli in modo compatto.

Per definire una variabile numerica a precisione singola, bisogna assegnare un nome seguito dal simbolo di dichiarazione !. È anche possibile non specificare nessun simbolo di dichiarazione, dato che la precisione singola è il tipo di variabile di default. Una variabile a precisione singola può memorizzare dei numeri, positivi o negativi, il cui valore assoluto è compreso tra 0 e $3{,}4*10^{38}$. I numeri il cui valore assoluto è inferiore a 10^{-45} vengono considerati uguali a zero.

Le variabili numeriche a precisione doppia hanno un nome seguito dal simbolo di dichiarazione # e possono memorizzare dei numeri, positivi e negativi, il cui valore assoluto è compreso tra 0 e $1{,}79*10^{308}$. I numeri il cui valore assoluto è inferiore a 10^{-324} vengono considerati uguali a zero.

Benché le variabili a virgola mobile possano contenere dei valori molto grandi e molto piccoli, sono limitate per quanto riguarda la precisione. Le variabili a precisione singola considerano fino a 6 cifre significative, mentre quelle a precisione doppia ne considerano 16.

Indipendentemente dalla loro precisione, QBasic converte i valori delle variabili a virgola mobile in un formato binario speciale. Un bit viene riservato per il segno della mantissa, altri bit sono utilizzati per contenere la mantissa stessa, un altro bit contiene il segno dell'esponente, e i bit rimanenti contengono il valore dell'esponente.

QBasic riporta i numeri in virgola mobile al formato standard in base 10, se possibile, o al formato scientifico menzionato in precedenza, nel caso in cui si tratti di numeri molto grossi, durante le operazioni di visualizzazione e stampa. La procedura di conversione tra la base 10 e la base 2 può a volte causare delle piccole alterazioni ai numeri. Ciò dipende dal fatto che non sempre si può rappresentare precisamente l'esatto valore della mantissa con il numero di bit ad essa assegnato. Le differenze, tuttavia, sono lievissime.

QBasic visualizza i valori delle variabili a virgola mobile in notazione scientifica o in notazione standard. In notazione scientifica, QBasic esprime ciascun numero come un numero moltiplicato per 10 elevato a potenza, dove 10 elevato a potenza viene scritto come E se si tratta di variabili a precisione singola, e come D nel caso di variabili a precisione doppia. Per esempio, si può assegnare un valore a una variabile a

precisione singola utilizzando l'istruzione seguente:

```
LET numero = 1,36E-3
```

Questo numero in formato decimale corrisponde a 0,00136. Il Programma 4.3 illustra questo concetto.

Si noti che tutti i programmi iniziano con un enunciato REM. Ciò viene fatto solamente per fornire alcune informazioni sul programma stesso, poiché QBasic ignora completamente questi enunciati. Si può aggiungere un'annotazione anche alla fine di una riga, facendo precedere il commento da un apostrofo (').

FUNZIONI NUMERICHE

Finora sono stati esaminati gli operatori aritmetici disponibili e i diversi tipi di variabile utilizzabili nei programmi. QBasic, tuttavia, dispone anche di una vasta gamma di funzioni numeriche predefinite che agiscono su valori numerici per generare nuovi valori. Alcune di queste funzioni sono riportate nella Tabella 4.2; l'Appendice B contiene l'elenco completo di tutte le funzioni incorporate in QBasic.

La funzione ABS può essere utilizzata per ottenere il valore assoluto di un numero. Il Programma 4.4 mostra l'uso della funzione ABS calcolando la differenza tra due età.

Programma 4.3 *Uso di variabili numeriche*

```
REM Uso di variabili numeriche [4-3]
num = .2
weightOfEarthInTons = 6588 * 10 ^ 21
balance = 1234567.89
balance# = 1234567.89
googol# = 1D+100             '1 * 10^100
PRINT num
PRINT weightOfEarthInTons
PRINT balance
PRINT balance#
PRINT 1 / googol#
PRINT googol# ^ 5
END

[Esecuzione]
.2
6.588E+24
1254568
1234567.89
ID-100
{ Overflow }
```

La funzione INT può essere usata per arrotondare dei numeri. Ad esempio, la funzione

```
INT(x + 0.5)
```

restituisce il valore di *x* arrotondato all'intero più vicino, mentre l'enunciato

```
INT(100 * x + 0.5) / 100
```

fornisce il valore di *x* arrotondato a due cifre decimali. Per arrotondare *x* a *n* cifre decimali, si sostituisca 100 nell'espressione sopra riportata con 1 (uno) seguito da *n* zeri. Quindi, per arrotondare un numero a 4 cifre decimali, si sostituisca 100 con 1000. Il Programma 4.5 illustra questa tecnica.

Tabella 4.2 *Alcune funzioni numeriche incorporate*

Funzione	Descrizione	Esempio
ABS(x)	valore assoluto di x	|x|
FIX(x)	parte intera di x	FIX(3.2) è uguale a 3
INT(x)	intero più vicino a x	INT(3.2) è uguale a 3
SGN(x)	segno di x (-1, 0, 1)	SGN(3.2) è uguale a 1

Programma 4.4 *Dimostrazione della funzione ABS*

```
REM Dimostrazione della funzione ABS [4-4]
INPUT "Età della prima persona: ", age1
INPUT "Età della seconda persona: ", age2
difference = ABS(age1 - age2)
PRINT "Ci sono"; difference; "anni di differenza."
END

[Esecuzione]
Età della prima persona: 7
Età della seconda persona: 10
Ci sono 3 anni di differenza.
```

Programma 4.5 *Arrotondamento di un numero*

```
REM Arrotondamento di un numero [4-5]
INPUT "Numero da arrotondare: "; x#
INPUT "Numero di posizioni decimali: "; n
roundedValue# = INT(x# * 10 ^ n + .5) / 10 ^ n
PRINT "Il valore arrotondato è "; roundedValue#
END
```

```
[Esecuzione]
Numero da arrotondare: -1.235
Numero di posizioni decimali: 2
Il valore arrotondato è -1.24
```

ESPRESSIONI NUMERICHE

Un'*espressione numerica* consiste di costanti, variabili, e/o funzioni numeriche combinate con operatori aritmetici. Le parentesi sono utili per determinare l'ordine con cui devono essere svolte le operazioni. QBasic calcola prima le espressioni racchiuse tra parentesi. Nel caso in cui ci siano delle parentesi nidificate, viene calcolata per prima l'espressione contenuta tra le parentesi più interne. A meno che non ci siano delle parentesi, la priorità degli operatori è la seguente:

esponenziale
negazione
moltiplicazione e divisione
divisione intera
MOD
addizione e sottrazione

Questa priorità degli operatori è quella normalmente utilizzata in matematica.

Tabella 4.3 *Priorità degli operatori*

Espressione	Interpretazione	Risultato
2+3*4	2+(3*4)	14
100 MOD 21/3	100 MOD (21/3)	2
-2^4	-(2^4)	-16
6*8/4+3	((6*8)/4)+3	15

QBasic calcola innanzi tutto gli elevamenti a potenza da sinistra verso destra, e quindi le negazioni seguendo la stessa direzione; prosegue quindi con la moltiplicazione e la divisione, sempre da sinistra a destra, quindi con la divisione intera e il calcolo del resto, e infine con l'addizione e la sottrazione. La Tabella 4.3 mostra alcuni esempi che illustrano la priorità degli operatori.

STRINGHE DI CARATTERI

Oltre che con i numeri, è ovviamente possibile lavorare con i caratteri. Dati di tipo carattere, tuttavia, devono essere memorizzati in *stringhe di caratteri*. Una stringa

può contenere una singola lettera, una parola, o intere frasi. Una *costante stringa* è una sequenza di caratteri. Nei programmi, le costanti stringa appaiono spesso racchiuse tra virgolette. La costante stringa "", che ha lunghezza uguale a zero, viene denominata *stringa nulla*. Le variabili che contengono delle costanti stringa seguono le stesse regole relative all'assegnazione dei nomi esaminate per le variabili numeriche, ma devono utilizzare il simbolo di dichiarazione $. Una costante stringa in QBasic può contenere fino a 32767 caratteri.

L'unico operatore disponibile per le stringhe è il simbolo +, che è l'operatore di concatenamento. Se a$ e b$ sono due stringhe, l'espressione a$+b$ genera una stringa composta dalla combinazione delle altre due. Per esempio, "mano" + "scritto" è uguale a "manoscritto".

Tabella 4.4 *Alcune funzioni stringa incorporate*

Funzione	Descrizione	Esempio
DATE$	data corrente	DATE$ potrebbe essere 22/10/1991
INSTR(a$,b$)	prima occorrenza di b$ in a$	INSTR("gatto","a") genera 1
LCASE$(a$)	converte a$ in minuscolo	LCASE$("Gatto") è "gatto"
LEFT$(a$,n)	n caratteri a sinistra di a$	LEFT$("gatto",2) è "ga"
LEN(a$)	numero di caratteri in a$	LEN("gatto") è 5
MID$(a$,m,n)	n caratteri di a$ partendo da m	MID$("gatto",2,1) è "a"
RIGHT$(a$,n)	n caratteri a destra di a$	RIGHT$("gatto",2) è "to"
UCASE$(a$)	converte a$ in maiuscolo	UCASE$("Gatto") è "GATTO"
VAL(a$)	a$ come numero	VAL("1991")+2 è 1993

FUNZIONI STRINGA

QBasic mette a disposizione numerose funzioni che richiedono come argomento una stringa di caratteri. Alcune delle funzioni stringa disponibili in QBasic sono riportate nella Tabella 4.4. Si faccia riferimento all'Appendice B per l'elenco completo di queste funzioni.

Il Programma 4.6 utilizza alcune funzioni stringa per estrarre il nome proprio dal nome completo di una persona, mentre il Programma 4.7 determina l'età della persona.

Nota: Perché la funzione DATE$ possa funzionare, l'orologio di sistema nel computer deve essere impostato correttamente.

Programma 4.6 *Estrazione del nome proprio*

```
REM Estrazione del nome proprio [4-6]
INPUT "Nome e cognome: ", fullName$
n = INSTR(fullName$, " ")      'Posizione del primo spazio
firstName$ = LEFT$(fullName$, n - 1)
PRINT "Il nome proprio è "; firstName$
END

[Esecuzione]
Nome e cognome: Mario Rossi
Il nome proprio è Mario
```

Programma 4.7 *Calcolo dell'età di una persona*

```
REM Calcolo dell'età di una persona [4-7]
INPUT "Data di nascita (mm/gg/aaaa): ", birthday$
age = VAL(RIGHT$(DATE$, 4)) - VAL(RIGHT$(birthday$, 4))
PRINT "Quest'anno hai"; age; "anni."
END

[Esecuzione]
Data di nascita (mm/gg/aaaa): 04/06/1937
Quest'anno hai 54 anni
```

ULTERIORI CONSIDERAZIONI SUI TIPI DI VARIABILE

Come detto in precedenza, un simbolo di dichiarazione (%, &, !, #, $) alla fine di un nome di variabile, specifica il tipo di variabile. Si dovrebbe inoltre notare che delle variabili con lo stesso nome, ma con un simbolo di dichiarazione differente, vengono considerate come variabili distinte, come mostrato dal Programma 4.8.

Sempre dal Programma 4.8 si può notare che si possono scrivere diversi enunciati sulla stessa riga purché vengano separati dal simbolo due punti (:). Tuttavia, alcuni affermano che questa non è una buona regola di programmazione, poiché rende il programma più difficile da interpretare.

Ci si ricordi che QBasic crea una variabile a precisione singola quando si omette il simbolo di dichiarazione, e la tratta come se fosse stato aggiunto il simbolo !. Il Programma 4.9 mostra che QBasic considera una variabile senza simbolo di dichiarazione uguale ad una cui è stato associato il simbolo !.

Programma 4.8 *Simboli differenti generano variabili diverse*

```
REM Simboli differenti generano variabili diverse [4-8]
a% = 5: a& = 987654321: a! = 2.5
a# = 3.14159265358979
a$ = "QBasic"
PRINT a%; a&
PRINT a!; a#
PRINT a$
END

[Esecuzione]
 5  987654321
 2.5  3.14159265358979
QBasic
```

Programma 4.9 *Variabili con e senza simbolo di dichiarazione*

```
REM Variabili con e senza simbolo di dichiarazione [4-9]
a = 2: b! = 3
PRINT a; a!; b; b!
END

[Esecuzione]
 2  2  3  3
```

OMISSIONE DEI SIMBOLI DI DICHIARAZIONE

Se a un certo punto ci si rende conto che l'uso dei simboli di dichiarazione risulta scomodo e poco pratico, si potrebbe avere la tentazione di ometterli e utilizzare sempre le variabili a precisione singola. Ciò può funzionare in alcuni casi, ma in altre situazione può rallentare il programma o non fornire una precisione adeguata nei calcoli. QBasic mette a disposizione una soluzione migliore tramite gli enunciati *DEFtipo*.

Se *x* è una qualsiasi lettera dell'alfabeto, l'enunciato

```
DEFINT x
```

specifica che tutte le variabili senza simbolo di dichiarazione il cui nome inizia con la lettera *x* devono essere definite come variabili intere. Se *x* e *z* sono due lettere dell'alfabeto, l'enunciato

```
DEFINT x-z
```

specifica che tutte le variabili senza simbolo di dichiarazione il cui nome inizia con una lettera compresa tra la *x* e la *z* devono essere definite come variabili intere. In generale, gli enunciati DEFINT possono contenere delle lettere, degli intervalli, o entrambi, separati da virgole.

Per esempio,

```
DEFINT R
```

significa che la variabile *reddito* sarà una variabile intera.

```
DEFINT R-T
```

significa che *reddito, salario* e *totali* saranno variabili intere, mentre *totale!* sarà una variabile a precisione singola, poiché ne è stato specificato il tipo.

Il Programma 4.10 mostra l'uso di DEFINT. Si noti che negli enunciati DEFINT appaiono solo lettere in maiuscolo. Se si utilizzano le minuscole, l'editor le converte automaticamente in maiuscole.

Programma 4.10 *Uso di DEFINT*

```
REM Uso di DEFINT [4-10]
DEFINT C, I, M-R
california = 23.7
illinois = 11.4
maryland# = 4.2
newYork = 17.6
texas = 14.2
PRINT california; illinois; maryland#; newYork; texas
END

[Esecuzione]
 24  11  4.199999809265137  18  14.2
```

Programma 4.11 *Confusione con le dichiarazioni*

```
REM Confusione con le dichiarazioni [4-11]
cost1 = 123.45        'Elaborata come cost1!
DEFDBL C
cost2 = 67.89         'Elaborata come cost2#
PRINT cost1; cost2    'Elaborata come cost1# e cost2#
PRINT cost1!
END

[Esecuzione]
 0  67.88999938964844
 123.45
```

Si noti il terzo valore visualizzato dal Programma 4.10. Il numero 4,2 diventa 4,199999809265137 durante la procedura di conversione da notazione standard a formato in virgola mobile e viceversa.

Si possono definire delle variabili senza simbolo di dichiarazione come intere lunghe, a precisione singola, a precisione doppia o stringa in modo analogo, utilizzando rispettivamente gli enunciati DEFLNG, DEFSNG, DEFDBL, e DEFSTR.

Benché gli enunciati DEFtipo possano apparire in qualsiasi punto del programma, la regola del buon programmatore raccomanda di utilizzare questa istruzione all'inizio del programma. In questo modo, si evitano degli errori come quelli che si verificano nel Programma 4.11 con la variabile *cost1*.

In questo caso, una variabile (COST1) appare come due variabili separate.

ARRAY

Le variabili finora esaminate risultano convenienti per memorizzare dei valori singoli. Alcune volte, tuttavia, capita di dover memorizzare una grossa quantità di dati tra loro relazionati. Situazioni di questo tipo possono essere gestite tramite l'uso degli array.

Un array è una serie di variabili dello stesso tipo che condividono lo stesso nome. Per far riferimento a ciascuna variabile dell'array, si utilizza un *indice* che indica la posizione occupata dalla variabile nell'array stesso. Le singole variabili vengono anche chiamate gli *elementi* dell'array.

Programma 4.12 *Uso di un array*

```
REM Uso di un array per memorizzare l'uomo dell'anno [4-12]
DIM manOfYear$(1979 TO 1982)
manOfYear$(1979) = "Ayatollah Khomeini"
manOfYear$(1980) = "Ronald Reagan"
manOfYear$(1981) = "Lech Walesa"
manOfYear$(1982) = "Il Computer"
INPUT "Inserire un anno compreso tra 1979 e 1982: ", year
PRINT manOfYear$(year); " è stato nominato uomo dell'anno."
END

[Esecuzione]
Inserire un anno compreso tra 1979 e 1982: 1982
Il Computer è stato nominato uomo dell'anno.
```

ARRAY A UNA DIMENSIONE

Gli array a una dimensione vengono utilizzati per contenere una lista di valori. Gli elementi in un array a una dimensione sono indicizzati da una sequenza di numeri interi. Se *nomeArray* è il nome di un array a una dimensione e i valori di indice sono compresi tra *M* e *N*, gli elementi dell'array possono essere scritti come *nomeArray(M)*, *nomeArray(M+1)*, e così via, fino a *nomeArray(N)*. Un array viene definito dall'enunciato

```
DIM nomeArray(M TO N)
```

Se si utilizza in un programma un array senza prima dichiararlo con l'istruzione DIM, vengono assegnate automaticamente 11 posizioni, con un indice compreso tra 0 e 10. Inoltre, un array con un indice compreso tra 0 e *N* può essere definito mediante l'enunciato

```
DIM nomeArray(N)
```

Ogni elemento di un array deve essere dello stesso tipo. Il tipo viene generalmente specificato tramite un simbolo di dichiarazione posto alla fine del nome dell'array. Il Programma 4.12 utilizza un array a una dimensione.

ARRAY A DUE DIMENSIONI

Se un array a una dimensione può essere utilizzato per memorizzare una lista di valori, un array a due dimensioni viene normalmente usato per contenere i valori di una tabella.

Tabella 4.5 *Statistiche economiche relative agli Stati Uniti*

	1981	1982	1983	1984
1. Cambio in percentuale del prezzo dei beni di consumo	10,4	6,1	3,2	4,3
2. Tasso di disoccupazione	7,5	9,5	9,5	7,4
3. Reddito medio familiare	25569	25216	25594	26433

Se le righe di questa tabella sono numerate da *M* a *N*, e le colonne da *S* a *T*, il valore che si trova nella riga *r* e nella colonna *c* della tabella verrà inserito nell'elemento *nomeArray(r,c)*. Un array di questo tipo viene definito tramite l'enunciato

```
DIM(M TO N, S TO T)
```

Il Programma 4.13 illustra l'uso di un array a due dimensioni accedendo ai dati della Tabella 4.5.

Programma 4.13 *Uso di un array a due dimensioni*

```
REM Uso di un array a due dimensioni [4-13]
DIM USstat#(1 TO 3, 1981 TO 1984)
USstat#(1, 1981) = 10.4
USstat#(1, 1982) = 6.1
USstat#(1, 1983) = 3.2
USstat#(1, 1984) = 4.3
USstat#(2, 1981) = 7.5
USstat#(2, 1982) = 9.5
USstat#(2, 1983) = 9.5
USstat#(2, 1984) = 7.4
USstat#(3, 1981) = 25569
USstat#(3, 1982) = 25216
USstat#(3, 1983) = 25594
USstat#(3, 1984) = 26433
INPUT "Cambio, disoccupazione o reddito (1, 2 o 3)"; category
INPUT "Anno compreso tra il 1981 e il 1984"; year
PRINT USstat#(category, year)
END

[Esecuzione]
Cambio, disoccupazione o reddito (1, 2 o 3)? 2
Anno compreso tra il 1981 e il 1984? 1983
 9.5
```

ARRAY A PIÙ DIMENSIONI

Si può definire un array con un massimo di 60 indici tramite l'enunciato

```
DIM(M TO N, S TO T....., U TO V)
```

I puntini di sospensione indicano l'eventuale presenza di altri indici. Un array di questo tipo consiste di (N-*M*+1)*(T-S+1)*....*(V-U+1) elementi.

Il nome di ogni singolo elemento ha il formato seguente:

```
nomeArray(a, b, ...., c)
```

dove *a* è compreso tra *M* e *N*, *b* tra *S* e *T*, e *c* tra *U* e *V*. Se si utilizza un array a due o più dimensioni senza prima definirlo con il comando DIM, viene assegnato l'intervallo 0-10 a ciascun indice. Inoltre, un array con il primo indice compreso tra 0 e *N*, il

secondo compreso tra 0 e *T*, e l'ultimo compreso tra 0 e *V*, può essere dichiarato con l'enunciato

```
DIM nomeArray(N, T, ..., V)
```

CONSIDERAZIONI SUGLI ARRAY E SULLA MEMORIA

Quando si definisce un array con il comando DIM, viene riservata una porzione di memoria per ciascun elemento dell'array. La memoria riservata per un array numerico è sufficiente per contenere tutti i valori che potrebbero essere assegnati ai vari elementi. Per contro, la memoria riservata per un array di tipo stringa non viene utilizzata per contenere le stringhe assegnate agli elementi, ma solo per memorizzare la lunghezza e la posizione della stringa stessa. Il numero massimo di elementi in ciascun array è 32768 per dimensione. QBasic non impone limiti al numero di array definibili; l'unica limitazione è imposta dalla memoria presente nel computer.

ARRAY STATICI E DINAMICI

L'istruzione DIM può specificare gli indici delle variabili tramite costanti o espressioni numeriche. Per esempio, gli enunciati DIM *nomeArray*(20) e DIM *nomeArray*(3 TO 100, 2 TO 17) utilizzano solo costanti, mentre gli enunciati DIM *nomeArray*(*a* TO *b*, 15) e DIM *nomeArray* (2**n*) utilizzano una o più espressioni.

QBasic assegna agli array delle porzioni di memoria in due modi differenti. Per gli array *statici* viene riservata una porzione di memoria nel momento in cui il compilatore elabora il programma. Ogni volta che si esegue il programma, questo spazio ha sempre la stessa dimensione e non può essere utilizzato per altri scopi. Agli array *dinamici*, invece, viene assegnata una porzione di memoria ogni volta che il programma viene eseguito. La dimensione di questo spazio varia ad ogni esecuzione e può essere liberato in qualsiasi momento.

Per default un array è statico, e diventa dinamico quando si verifica una delle condizioni seguenti:

1. l'array viene definito utilizzando una o più variabili per specificare l'indice. Per esempio,

   ```
   DIM nomeArray(x)
   ```

2. appare l'enunciato REM $DYNAMIC prima del dimensionamento dell'array.

Nota: Gli enunciati composti dalla parola chiave REM seguita da una delle parole riservate $DYNAMIC, $STATIC e $INCLUDE, sono denominati metacomandi e

forniscono istruzioni particolari al compilatore. In questo caso particolare, REM $DYNAMIC indica al compilatore di rendere dinamici gli array successivi.

Un array dinamico può essere rimosso completamente dalla memoria tramite il comando ERASE. In particolare, l'enunciato

```
ERASE nomeArray
```

libera la porzione di memoria precedentemente riservata per l'array specificato. Dopo aver cancellato un array dinamico, è possibile ridimensionarlo con indici differenti. Tuttavia, il numero delle dimensioni deve restare uguale a quello precedente. Ciò è dovuto al fatto che l'enunciato usato per definire l'array (ad esempio DIM *nomeArray*(x)) non cambia.

Se si utilizza il comando ERASE con una variabile statica, la porzione di memoria da essa occupata non viene liberata; tuttavia, vengono cancellati tutti i valori assegnati ai vari elementi. I valori numerici diventano zero e i valori stringa diventano stringhe nulle.

Gli array dinamici sono più flessibili di quelli statici. Un array statico ha sempre la stessa dimensione e si deve cambiare il programma per modificarlo. La dimensione di un array dinamico, invece, può essere determinata durante l'esecuzione del programma.

Inoltre, gli array dinamici sono più efficienti per quanto riguarda l'uso della memoria. Lo spazio da loro occupato, infatti, può essere liberato in qualsiasi momento ed essere utilizzato per altri scopi. Gli array statici, tuttavia, dispongono di un'importante caratteristica che ha giustificato la loro implementazione in QBasic: consentono un accesso più rapido.

Programma 4.14 *Array statici e dinamici*

```
REM Array statici e dinamici [4-14]
DIM a$(1 TO 25)                        'Array statico
INPUT "Dimensione dell'array:"; n      'Dimensione dell'array
DIM c(n)                               'Array dinamico
REM $DYNAMIC
DIM d(17)                              'Array dinamico
REM $STATIC                            'Nuovo default
DIM b(50)                              'Array statico
ERASE d
DIM d(20 TO 50)                        'Array dinamico
a$(22) = "ABC"
ERASE a$
PRINT a$(22) + "DEF"
END

[Esecuzione]
Dimensione dell'array: 5
DEF
```

INSERIMENTO DEI DATI

Ora che si è visto come memorizzare i dati nella memoria del computer e come svolgere alcune operazioni con le variabili e gli array, si esamineranno i comandi necessari per fornire ai programmi i dati da elaborare. QBasic mette a disposizione diversi comandi per l'inserimento dei dati, tra cui LET, INPUT, READ/DATA e INPUT$, che verranno discussi nelle sezioni seguenti.

LET E INPUT

I dati possono essere memorizzati nelle variabili in diversi modi. Gli enunciati LET e INPUT, che sono già stati utilizzati in questo libro, sono quelli maggiormente conosciuti. L'enunciato

```
LET variabile = espressione
```

o in forma abbreviata

```
variabile = espressione
```

assegna il valore dell'espressione alla variabile. Per esempio,

```
LET pi = 3,14159
```

inserisce il valore 3,14159 nella variabile *pi*, mentre

```
LET nome$ = "Rossi"
```

inserisce il nome "Rossi" nella variabile *nome$*.

Il valore e la variabile devono essere dello stesso tipo. Se si cerca di inserire un valore stringa in una variabile numerica o viceversa, viene generato un messaggio di errore. Se due valori sono entrambi numerici, ma di tipo differente, il valore viene convertito nel tipo della variabile. Per esempio,

```
età% = 3.2
```

assegna il valore 3 (arrotondato all'intero più vicino) alla variabile intera età%.

LET viene utilizzato per inserire dei valori all'interno di un programma. È tuttavia necessario un altro comando che consenta di prelevare i dati dall'esterno, richiedendoli all'utente. L'enunciato

```
INPUT variabile
```

visualizza sullo schermo un punto di domanda e attende che l'utente digiti una risposta che viene quindi memorizzata nella variabile. La variante

```
INPUT "richiesta"; variabile
```

visualizza il messaggio specificato prima del punto di domanda. Se il punto e virgola in questo enunciato viene sostituito da una virgola, il messaggio viene visualizzato senza il punto di domanda. L'enunciato

```
INPUT "richiesta"; variabile1, variabile2, ...
```

consente di richiedere all'utente più informazioni e di memorizzarle in diverse variabili. Per esempio, l'enunciato

```
INPUT "Specifica la tua età e il tuo peso: ", età%, peso%
```

visualizza la richiesta

```
Specifica la tua età e il tuo peso:
```

senza il punto di domanda alla fine, attende che l'utente inserisca due valori separati da un virgola e che prema Invio. I valori vengono quindi inseriti nella variabili intere età% e peso%.

READ E DATA

L'enunciato DATA elenca dei valori che vengono assegnati a delle variabili dal comando READ. Per esempio, un enunciato nella forma

```
READ x
```

indica al computer di localizzare la prima serie di dati non assegnati specificati con un comando DATA e di memorizzare il primo valore di questa serie nella variabile *x*. Una serie di enunciati come

```
READ x
READ y
READ z
```

possono essere sostituiti da un unico enunciato

```
READ x, y, z
```

Ogni enunciato DATA contiene una o più costanti separate da virgole. Generalmente, ogni enunciato DATA contiene diversi elementi relazionati. È una buona regola

disporre tutti gli enunciati DATA alla fine del programma e farli precedere da un comando REM che fornisca delle delucidazioni sui dati contenuti in questi enunciati, come mostrato nel Programma 4.15.

INPUT$

Se *a$* è una variabile stringa e *n* un numero intero compreso tra 1 e 32767, l'enunciato

```
a$ = INPUT$(n)
```

indica al programma di interrompere l'esecuzione fino a quando l'utente non ha premuto *n* caratteri dalla tastiera. Quindi, la stringa composta dagli *n* caratteri digitati viene memorizzata nella variabile *a$* e il programma continua con la riga successiva.

La funzione INPUT$(*n*) con *n*=1, viene spesso utilizzata per richiedere una selezione da menu. A differenza di quanto accade con il comando INPUT, in questo caso la lettera digitata non appare sullo schermo e non è necessario premere Invio per confermare la risposta. Il Programma 4.16 illustra questa tecnica.

Programma 4.15 *Gli enunciati READ e DATA*

```
REM Gli enunciati READ e DATA [4-15]
READ person$, yearOfBirth
PRINT "Quando negli Stati Uniti è nata la costituzione,"
PRINT person$; " aveva"; 1776 - yearOfBirth; "anni."
READ person$
READ yearOfBirth
PRINT "Quando è scoppiata la guerra civile, "
PRINT person$; " aveva"; 1861 - yearOfBirth; "anni."
REM --- Dati: Persona, anno di nascita
DATA Thomas Jefferson, 1743
DATA Abraham Lincoln, 1809
END

[Esecuzione]
Quando negli Stati Uniti è nata la costituzione,
Thomas Jefferson aveva 33 anni.
Quando è scoppiata la guerra civile,
Abraham Lincoln aveva 52 anni.
```

Programma 4.16 *La funzione INPUT$*

```
REM La funzione INPUT$ [4-16]
PRINT "Digita una lettera."
letter$ = INPUT$(1)
PRINT "La lettera digitata è "; letter$
END

[Esecuzione]
Digita una lettera.                 {Si digiti la lettera T}
La lettera digitata è T
```

TIPI DI DATI NON CORRISPONDENTI

Tutti gli enunciati di inserimento sopra riportati assegnano una costante a una variabile di tipo specificato. Nel caso che la costante non sia dello stesso tipo della variabile, QBasic cerca di adattare la costante nel migliore dei modi.

Se si tratta di una variabile intera e di una costante a singola o doppia precisione, QBasic arrotonda la costante, se possibile, prima di assegnarla alla variabile. Un esempio di un numero che non può essere arrotondato è *n%*=41.234,56. Questo enunciato genererebbe un errore di overflow dato che una variabile intera non può contenere un valore maggiore di 32000. Se si cerca di assegnare una costante stringa a una variabile numerica, viene generato il messaggio 'Tipi di dati non corrispondenti'. I valori numerici e quelli di tipo stringa sono totalmente incompatibili.

In un enunciato LET, il valore a destra del segno uguale può essere sia una costante che un'espressione. Se questo valore è un'espressione numerica e la variabile a sinistra del segno uguale è una variabile stringa, viene visualizzato il messaggio di errore 'Tipi di dati non corrispondenti'. Se invece si assegna una costante numerica a una variabile stringa con un qualsiasi enunciato di inserimento, QBasic inserisce il numero come stringa.

VISUALIZZAZIONE DEI DATI SULLO SCHERMO

Si è visto come inserire dei dati nel computer e come manipolarli. Si vedrà ora come prelevare questi dati e come visualizzarli sullo schermo. Il comando PRINT, usato in combinazione con le istruzioni TAB e LOCATE, può visualizzare dei dati in qualunque posizione dello schermo. Esiste inoltre una variante del comando PRINT, PRINT USING, che consente di personalizzare la visualizzazione dei dati impostando il formato desiderato tramite l'uso di apposite maschere.

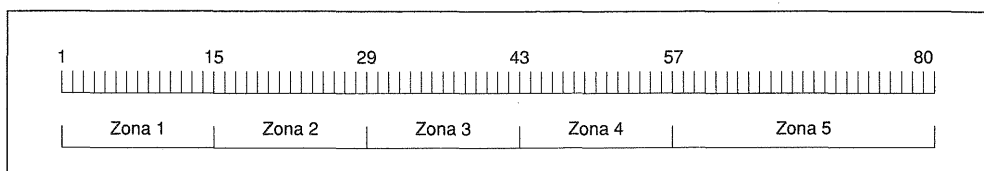



Figura 4.1 *Le zone del comando PRINT*

ZONE DEL COMANDO PRINT

Lo schermo può contenere 25 o più righe di testo, ciascuna delle quali composta da 80 caratteri. Si dovrebbero immaginare le righe suddivise in cinque zone, come quelle mostrate in Figura 4.1.

Quando l'enunciato PRINT è seguito da diversi elementi separati da un punto e virgola, QBasic li visualizza uno dopo l'altro. Se vengono utilizzate le virgole al posto dei punti e virgola, QBasic li visualizza in zone consecutive. Il Programma 4.17 utilizza le zone del comando PRINT per generare una tabella che mostri la media delle spese annuali per determinati beni di consumo. Si noti che la virgola iniziale nel primo comando PRINT di questo programma viene utilizzata per saltare la prima zona, in modo che le intestazioni siano allineate con i numeri.

Nota: Se si dispone di una scheda grafica come la CGA, la EGA o la VGA, si può passare alla modalità di visualizzazione a 40 colonne tramite il comando

```
WIDTH 40
```

e ritornare alla modalità standard con

```
WIDTH 80
```

In modalità a 40 colonne, ci sono solamente due zone per il comando PRINT, una di 14 caratteri e l'altra di 26.

IL COMANDO TAB

Mentre le zone del comando PRINT consentono di organizzare i dati in colonne separate tra loro da 14 posizioni, il comando TAB consente di accedere ad ogni posizione di ciascuna riga. Se un elemento in un enunciato PRINT è preceduto da

```
TAB(n)
```

dove *n* è un numero intero compreso tra 1 e 80 (oppure 1 e 40 se ci si trova in modalità

40 colonne), QBasic visualizza quell'elemento partendo dalla posizione *n* della riga corrente. Il Programma 4.18 illustra l'uso del comando TAB.

Programma 4.17 *Zone del comando PRINT*

```
REM Zone del comando PRINT [4-17]
PRINT , "Cibo", "Casa", "Trasporti"
PRINT "Massimo 20%", 4838, 10188, 6949
PRINT "Medio 20%", 2877, 5032, 3451
PRINT "Minimo 20%", 1753, 1730, 1231
END

[Esecuzione]
                Cibo          Casa           Trasporti
Massimo 20%     4838          10188          6949
Medio 20%       2877          5032           3451
Minimo 20%      1753          1730           1231
```

Programma 4.18 *Il comando TAB*

```
REM Il comando TAB [4-18]
PRINT TAB(17); "Cibo"; TAB(26); "Casa"; TAB(37); "Trasporti"
PRINT "Massimo 20%"; TAB(16); 4838; TAB(25); 10188; TAB(36); 6949
PRINT "Medio 20%"; TAB(16); 2877; TAB(25); 5032; TAB(36); 3451
PRINT "Minimo 20%"; TAB(16); 1753; TAB(25); 1730; TAB(36); 1231
END

[Esecuzione]
                Cibo      Casa       Trasporti
Massimo 20%     4838      10188      6949
Medio 20%       2877      5032       3451
Minimo 20%      1753      1730       1231
```

Programma 4.19 *Visualizzazione di un messaggio al centro dello schermo*

```
REM Un messaggio al centro dello schermo [4-19]
a$ = "Buon compleanno"
CLS
LOCATE 12, (80 - LEN(a$)) \ 2
PRINT a$
END
```

LOCATE

Per quanto riguarda la visualizzazione del testo, lo schermo è suddiviso in 25 o più righe orizzontali (numerate 1, 2, 3 eccetera) e 80 (o 40) colonne verticali (numerate

1, 2, 3, eccetera). L'enunciato

```
LOCATE r, c
```

sposta il cursore sulla riga *r* e sulla colonna *c* dello schermo. Il comando PRINT successivo visualizzerà i dati specificati partendo da quella posizione. Il Programma 4.19 visualizza la stringa 'Buon compleanno' al centro di uno schermo in modalità 80 colonne. Il comando CLS pulisce lo schermo, e quindi l'istruzione LOCATE sposta il cursore sulla dodicesima riga e determina la posizione orizzontale sottraendo la lunghezza della stringa da visualizzare dal valore 80 e dividendo il risultato per due.

PRINT USING

per editing

L'enunciato PRINT USING viene utilizzato per visualizzare dei dati numerici in un formato personalizzato, aggiungendo, ad esempio, le virgole come separatori delle migliaia e un simbolo valutario all'inizio del numero stesso. Il comando PRINT USING richiede una stringa, denominata stringa di formato, che specifichi il formato di visualizzazione.

Una stringa di formato tipica è "##,###,###.##", ed è composta da 13 caratteri. L'enunciato

```
PRINT USING "##,###,###.##"; n
```

riserva 13 posizioni (un campo) per visualizzare il numero *n*. QBasic visualizza il numero giustificandolo a destra, arrotondandolo a due posizioni decimali, e inserendo le virgole di separazione delle migliaia dopo ogni gruppo di tre cifre a sinistra del punto decimale. La Figura 4.2 mostra tre possibili valori di *n* e la relativa visualizzazione.

n	*Visualizzazione*
1234	1,234.00
12.345	12.35
1234567.89	1,234,567.89

Figura 4.2 *Effetto di PRINT USING "##,###,###.##"*

Se il primo carattere in una stringa di formato è il segno dollaro, QBasic visualizza il numero con quel simbolo posto all'estrema sinistra del campo. Se i primi due caratteri sono due segni del dollaro, QBasic visualizza il numero ponendo il simbolo $ davanti alla prima cifra. Si veda la Figura 4.3.

Gli enunciati PRINT USING consentono di combinare del testo con dei numeri in diversi modi. Si può ad esempio includere del testo all'interno di una stringa di formato insieme agli speciali caratteri di formattazione. In un caso del genere, QBasic visualizza il testo esattamente come appare, e mostra i numeri con il formato specificato. Il Programma 4.20 mostra un esempio di questo metodo.

Il carattere barra rovesciata, \, può essere utilizzato per formattare del testo con una stringa di formato. Se *c$* è una stringa, l'enunciato

```
PRINT USING "\           \"; c$
```

visualizza i primi *n* caratteri di *c$*, dove *n* rappresenta la lunghezza della stringa di formato "\   \". Ciò significa che ci sono *n*-2 spazi vuoti tra le due barre rovesciate. Il punto esclamativo, !, può essere usato per estrarre il primo carattere da una stringa. La Figura 4.4 mostra degli esempi che mostrano l'uso della barra rovesciata e del punto esclamativo.

Enunciato	*Visualizzazione*
PRINT USING "$####.##"; 45.78	$ 45.78
PRINT USING "$$###.##"; 45.78	$45.78

Figura 4.3 *Visualizzazione del segno $ con PRINT USING*

Programma 4.20 *Combinazione di testo e formato in PRINT USING*

```
REM  Combinazione di testo e formato in PRINT USING [4-20]
INPUT "Capitale: ", principal
a$ = "Il bilancio dopo un anno è di $$###,###.##"
PRINT USING a$; 1.06 * principal
END

[Esecuzione]
Capitale: 1234.56
Il bilancio dopo un anno è di    $1,308.63
```

Il Programma 4.21 mostra che una stringa di formato può contenere alcuni gruppi di caratteri per la formattazione delle stringhe e per la formattazione dei numeri.

L'enunciato PRINT USING viene spesso usato per incolonnare dei numeri. Per esempio, si esamini la colonna centrale dell'output generato dal programma 4.18. L'ideale sarebbe che i numeri fossero allineati a destra , in modo che la cifra più a destra appaia nella stessa colonna. Il Programma 4.22 utilizza il comando PRINT USING per raggiungere questo obiettivo e migliorare la leggibilità aggiungendo delle virgole nei punti appropriati.

La Tabella 4.6, che mostra il risultato ottenuto dall'enunciato PRINT USING *a$; n*, contiene alcuni simboli addizionali che possono essere usati nelle stringhe di formato.

Enunciato	*Visualizzazione*
PRINT USING "\\smos"; "Computer"	Cosmos
PRINT USING "\ \"; "Computer"	Compu
PRINT USING "!"; "Computer"	C

Figura 4.4 *Caratteri per la formattazione delle stringhe*

Programma 4.21 *Alcune caratteristiche del comando PRINT USING*

```
REM Alcune caratteristiche del comando PRINT USING [4-21]
a$ = "!!! ha venduto ###,### PC in \ \ ####"
PRINT USING a$; "International"; "Business"; "Machines"; 158000;
"Dicembre"; 1984
END

[Esecuzione]
IBM ha venduto 158,000 PC in Dic 1984
```

Il carattere & può essere utilizzato in una stringa di formato per visualizzare un'intera stringa. Se un singolo carattere di formattazione, come # o \, viene preceduto dal simbolo di sottolineatura, _, il carattere perde la sua funzione e viene visualizzato normalmente. Il Programma 4.23 illustra l'uso dei caratteri & e _.

Programma 4.22 *Uso di PRINT USING per migliorare la leggibilità*

```
REM Uso di PRINT USING per migliorare la leggibilità [4-22]
PRINT "                Cibo      Casa     Trasporti"
a$ = "\         \    #,###    ##,###      #,###"
PRINT USING a$; "Massimo 20%"; 4838; 10188; 6949
PRINT USING a$; "Medio 20%"; 2877; 5032; 3451
PRINT USING a$; "Minimo 20%"; 1753; 1730; 1231
END

[Esecuzione]
                Cibo       Casa       Trasporti
Massimo 20%     4,838      10,188     6,949
Medio 20%       2,877      5,032      3,451
Minimo 20%      1,753      1,730      1,231
```

Tabella 4.6 *Caratteri di formattazione addizionali per PRINT USING*

Simbolo(i)	Descrizione	n	a$	Risultato
**	Inserisce degli asterischi al posto degli spazi in testa	23	"**###"	***23
*	Visualizza un asterisco come primo carattere del campo	23	"####"	* 23
^^^^	Visualizza il numero in formato esponenziale	23	"#.##^^^^"	2.30E+001
+	Riserva uno spazio per il segno del numero	-23	"###-"	23-

EFFETTI SPECIALI SU VIDEO MONOCROMATICO CON IL COMANDO COLOR

A seconda del tipo di monitor utilizzato, il testo può essere visualizzato con effetti speciali, ad esempio sottolineato o lampeggiante, oppure a colori. Un video monocromatico è un tipo particolare di monitor che può essere collegato a un PC IBM o compatibile. I sistemi monocromatici sono ideali per la visualizzazione del testo, ma sono limitati a due colori: il bianco e il nero (alcuni monitor monocromatici utilizzano il nero e il verde o il nero e l'ambra). Per ogni carattere che viene visualizzato sullo schermo, si parla di colore di primo piano per indicare il colore del carattere stesso, e di colore di sfondo per indicare il colore dell'area che circonda il carattere.

Il comando COLOR può essere utilizzato per visualizzare del testo sottolineato, lampeggiante, in grassetto o in inverso su video monocromatici. La Tabella 4.7 mostra tutte le combinazioni di attributi possibili e gli enunciati COLOR necessari per generarle.

Dopo aver impartito un comando COLOR, tutto il testo successivamente visualizzato viene rappresentato con gli attributi specificati. Si possono utilizzare effetti differenti nella stessa videata. Il Programma 4.24 mostra i diversi effetti disponibili con i sistemi monocromatici.

Programma 4.23 *I simboli & e _ nel comando PRINT USING*

```
REM I simboli & e _ nel comando PRINT USING [4-23]
b$ = "un megabyte"
PRINT USING "La directory _\GIOCHI occupa &."; b$
END

[Esecuzione]
La directory \GIOCHI occupa un megabyte.
```

Programma 4.24 *Alcuni effetti su video monocromatici*

```
REM Alcuni effetti su video monocromatici [4-24]
COLOR 17, 0
PRINT "Queste lettere sono sottolineate e lampeggianti."
COLOR 0, 7
PRINT "Le lettere in questa riga sono in inverso."
COLOR 0, 0
PRINT "Questa riga è illeggibile."
COLOR 7, 0
PRINT "Questa è una visualizzazione standard."
END
```

Tabella 4.7 *Effetti speciali per la visualizzazione del testo su monitor monocromatici*

Primo piano	Sfondo	Comando
bianco	nero	COLOR 7, 0
bianco, sottolineato	nero	COLOR 1, 0
bianco, lampeggiante	nero	COLOR 23, 0
bianco, sottolineato, lampeggiante	nero	COLOR 17, 0
bianco intenso	nero	COLOR 15, 0
bianco intenso, sottolineato	nero	COLOR 9, 0
bianco intenso, lampeggiante	nero	COLOR 31, 0
bianco intenso, sottolineato, lampeggiante	nero	COLOR 25, 0
nero	bianco	COLOR 0, 7
nero, lampeggiante	nero	COLOR 16, 0
nero, lampeggiante	bianco	COLOR 16, 7
nero	nero	COLOR 0, 0

VISUALIZZAZIONE SU MONITOR A COLORI

La maggior parte dei monitor a colori sono collegati a schede grafiche CGA, EGA o VGA. Se si dispone di una di queste schede, il testo può essere visualizzato scegliendo uno dei 16 colori disponibili, riportati nella Figura 4.5, per il colore di primo piano e per quello di sfondo.

```
0  Nero       4  Rosso      8  Grigio          12 Rosso chiaro
1  Blu        5  Magenta    9  Blu chiaro      13 Magenta chiaro
2  Verde      6  Marrone    10 Verde chiaro    14 Giallo
3  Ciano      7  Bianco     11 Ciano chiaro    15 Bianco intenso
```

Figura 4.5 *I colori disponibili*

Inoltre, il colore di primo piano può essere reso lampeggiante. Se si dispone di una scheda EGA o VGA, sono disponibili 48 colori addizionali.

Se *f* è un numero compreso tra 0 e 15 e *b* un numero compreso tra 0 e 7, l'enunciato

```
COLOR f, b
```

indica al programma di visualizzare qualsiasi carattere successivamente riportato a video con il colore di primo piano *f* e il colore di sfondo *b* (il carattere stesso avrà colore *f* e il piccolo rettangolo sullo schermo che contiene il carattere avrà colore *b*). I caratteri visualizzati sullo schermo prima dell'esecuzione di questo comando resteranno inalterati. Se si aggiunge 16 al numero *f*, si ottiene un colore di primo piano lampeggiante. Il Programma 4.25 mostra l'uso del comando COLOR.

Programma 4.25 *Uso dei colori*

```
REM Uso dei colori [4-25]
COLOR 14, 4
PRINT "Le lettere in questa riga sono gialle su sfondo rosso."
COLOR 30, 4
PRINT "Questa riga è uguale alla prima, ma è lampeggiante."
COLOR 7, 0
PRINT "Questa è una visualizzazione standard."
END
```

VISUALIZZAZIONE A COLORI SU MONITOR EGA O VGA

I monitor che supportano la risoluzione EGA o VGA sono dei monitor particolari in grado di visualizzare 64 colori in modalità di testo. Questi colori sono numerati da 0 a 63. I colori blu, verde, ciano, rosso, magenta e giallo sono disponibili in sei sfumature differenti, e ci sono inoltre quattro livelli di grigi. La Tabella 4.8 riporta i numeri associati ai colori di ciascun gruppo. Come regola generale, si tenga presente che in ciascun gruppo il numero più alto corrisponde al colore più chiaro e/o intenso. Al marrone è associato il numero 20. I colori associati ai 23 numeri rimanenti sono difficili da definire. Il Programma 4.26 visualizza tutti i 64 colori. Questo programma utilizza dei comandi che non sono stati ancora spiegati e che risulteranno chiari in seguito.

Tabella 4.8 *Sfumature dei colori*

Colore	Sfumature
Blu	1, 8, 9, 17, 25, 41, 57
Verdi	2, 10, 16, 18, 24, 26, 34, 42, 48, 58
Ciani	3, 11, 27, 31, 35, 43,59
Rossi	4, 12, 32, 36, 44, 52, 53, 60
Magenta	5, 13, 21, 29, 45, 47, 61
Gialli	6, 14, 30, 54, 62
Grigi	0 (nero), 7, 15, 56, 63 (bianco intenso)

QBasic usa due serie di numeri per controllare la scelta dei colori. La prima consiste dei numeri compresi tra 0 e 63 che identificano i 64 colori differenti.

La Figura 4.6 mostra 16 barattoli di vernice etichettati da 0 a 15. QBasic riserva per ogni barattolo una locazione di memoria che tiene traccia del colore in esso contenuto. L'elenco dei barattoli e dei colori viene chiamata *palette*, che in inglese significa tavolozza. Il colore in ciascun barattolo può essere modificato in qualsiasi momento. Se *m* è un numero compreso tra 0 e 15 e *c* un altro numero compreso tra 0 e 63, l'enunciato

```
PALETTE m, c
```

sostituisce il colore che si trova nel barattolo *m* con il colore *c*. I colori di default (cioè i colori che si trovano nei barattoli prima dell'esecuzione di un comando PALETTE) sono riportati nella Tabella 4.9. Si noti che i numeri compresi tra 0 e 15 che identificano i barattoli vengono spesso chiamati *attributi*.

Programma 4.26 *I 64 colori disponibili con le schede EGA e VGA*

```
REM I 64 colori disponibili con le schede EGA e VGA [4-26]
DEFINT I-J
FOR i = 0 TO 63 STEP 8
  CLS
  FOR j = 0 TO 7
    PALETTE j + 1, i + j
    COLOR j + 1, 0
    PRINT USING "Colore ##"; i + j
    PRINT
  NEXT j
  SLEEP 4               'Pausa di 4 secondi
NEXT i
END
```

Se *f* e *b* sono dei numeri compresi rispettivamente tra 0 e 15 e 0 e 7, l'enunciato

```
COLOR f, b
```

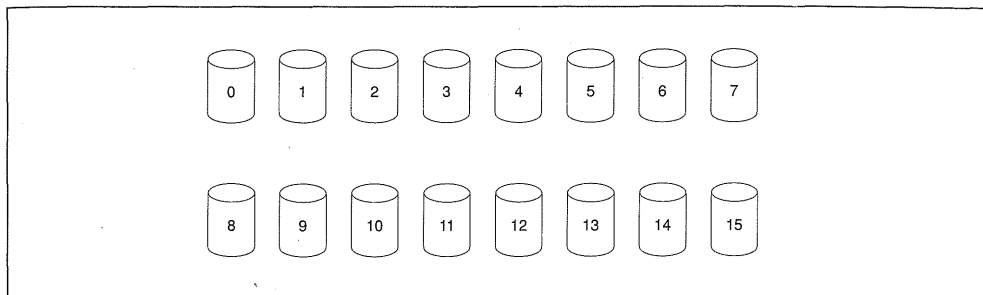



Figura 4.6 *Una palette*

indica al programma di visualizzare tutti i caratteri successivi usando come colore di primo piano quello che si trova nel barattolo *f*, e come colore di sfondo quello che si trova nel barattolo *b*. Si tenga presente che la scheda grafica non memorizza i due colori per ciascuna posizione dello schermo, ma solamente i numeri dei due barattoli.

Il monitor esamina continuamente la palette per determinare i colori da utilizzare. I caratteri visualizzati sullo schermo prima dell'esecuzione di un comando COLOR conservano i colori originali. Tuttavia, ogni volta che si esegue un comando PALETTE, QBasic cambia la rappresentazione di tutti i caratteri i cui colori di primo piano e di sfondo appartengono al barattolo a cui è stato cambiato il colore.

Tabella 4.9 *La palette di default*

Barattolo	0,	1,	2,	3,	4,	5,	6,	7	8,	9,	10,	11,	12,	13,	14,	15
Colore	0,	1,	2,	3,	4,	5,	20,	7,	56,	57,	58,	59,	60,	61,	62,	63

Programma 4.27 *Uso dei colori su monitor EGA o VGA*

```
REM Uso dei colori su monitor EGA o VGA [4-27]
COLOR 6, 7
PRINT "primo piano marrone, sfondo bianco"
PALETTE 12, 1
COLOR 12, 7
PRINT "primo piano blu, sfondo bianco"
SLEEP 5                              'Pausa di 5 secondi
PALETTE 7, 4
PRINT "Tutti i caratteri hanno uno sfondo rosso."
END
```

Nello stesso momento possono apparire al massimo 16 colori differenti sullo schermo. Il Programma 4.27 mostra l'uso degli enunciati COLOR e PALETTE con monitor EGA o VGA.

INVIO DI DATI A UNA STAMPANTE

Come si è visto, ci sono diverse opzioni per visualizzare i dati sullo schermo. Il video, tuttavia, non è l'unica periferica di output; si potrebbero infatti voler inviare i dati ad una stampante. A questo scopo, si può utilizzare il comando LPRINT che funziona esattamente come l'enunciato PRINT ad eccezione del fatto che l'output avviene su stampante.

Tabella 4.10 *Codici di controllo per stampanti a matrice di punti IBM, Epson e compatibili*

Codice	Effetto
`LPRINT CHR$(7);`	Suona il cicalino
`LPRINT CHR$(12);`	Avanzamento pagina
`LPRINT CHR$(13);`	Ritorno a capo e avanzamento riga
`LPRINT CHR$(14);`	Attiva la modalità doppia larghezza fino al primo avanzamento riga o comando LPRINT CHR$(20)
`LPRINT CHR$(15);`	Attiva la modalità compressa
`LPRINT CHR$(18);`	Disattiva la modalità compressa
`LPRINT CHR$(20);`	Disattiva la modalità larghezza doppia
`LPRINT CHR$(27); "-"; CHR$(1)`	Attiva il sottolineato
`LPRINT CHR$(27); "-"; CHR$(0)`	Disattiva il sottolineato
`LPRINT CHR$(27); "0";`	Stampa 8 righe per pollice
`LPRINT CHR$(27); "1";`	Stampa 10 righe per pollice
`LPRINT CHR$(27); "2";`	Stampa 6 righe per pollice*
`LPRINT CHR$(27); "4";`	Attiva il corsivo*
`LPRINT CHR$(27); "5";`	Disattiva il corsivo*
`LPRINT CHR$(27); "@";`	Ripristina le impostazioni di default*
`LPRINT CHR$(27); "E";`	Attiva la modalità enfatizzata
`LPRINT CHR$(27); "F";`	Disattiva la modalità enfatizzata
`LPRINT CHR$(27); "G";`	Attiva la doppia sottolineatura
`LPRINT CHR$(27); "H";`	Disattiva la doppia sottolineatura
`LPRINT CHR$(27); "S"; CHR$(1)`	Attiva apice
`LPRINT CHR$(27); "S"; CHR$(0)`	Attiva indice
`LPRINT CHR$(27); "T";`	Disattiva apice e indice

*Solo stampanti Epson e compatibili

I punti e virgola sopprimono la combinazione ritorno a capo/avanzamento riga, le virgole formattano i dati in zone, la funzione TAB posiziona i dati in un punto specifico della riga, e LPRINT USING opera esattamente come PRINT USING. Inoltre, LPRINT consente di controllare la dimensione dei caratteri, il font, la spaziatura tra le righe e la lunghezza della pagina. La Tabella 4.10 riporta i codici per il controllo della stampante che si adattano alla maggior parte delle stampanti a matrice di punti. Si faccia attenzione a distinguere la lettera l (elle) dal numero 1 (uno). Il comando WIDTH specifica il massimo numero di caratteri che deve essere stampato su ciascuna riga. QBasic aggiunge automaticamente un ritorno a capo e un avanzamento riga. Se *n* è un numero positivo intero, l'enunciato

```
WIDTH "LPT1:", n
```

indica che ciascuna riga deve contenere al massimo *n* caratteri. L'impostazione di default prevede 80 caratteri per riga.

Tabella 4.11 *Codici di controllo per stampanti LaserJet Hewlett-Packard e compatibili*

Codice*	Effetto
`LPRINT CHR$(27); "&l3D";`	Stampa 3 righe per pollice
`LPRINT CHR$(27); "&l4D";`	Stampa 4 righe per pollice
`LPRINT CHR$(27); "&l6D";`	Stampa 6 righe per pollice
`LPRINT CHR$(27); "&lnD";`	Imposta la lunghezza pagina su n righe
`LPRINT CHR$(27); "&lnE";`	Imposta il margine alto su n righe
`LPRINT CHR$(27); "&lnL";`	Imposta il margine sinistro sulla colonna n
`LPRINT CHR$(27); "&lnR";`	Sposta il cursore sulla riga n
`LPRINT CHR$(27); "&lnC";`	Sposta il cursore sulla colonna n
`LPRINT CHR$(27); "&ldD";`	Attiva il sottolineato
`LPRINT CHR$(27); "&ld@";`	Disattiva il sottolineato
`LPRINT CHR$(27); "(s0P";`	Spaziatura fissa
`LPRINT CHR$(27); "(s1P";`	Spaziatura proporzionale
`LPRINT CHR$(27); "(snH";`	Passo di *n* caratteri/pollice
`LPRINT CHR$(27); "(snV";`	Carattere di altezza *n* punti
`LPRINT CHR$(27); "(s0S";`	Carattere verticale
`LPRINT CHR$(27); "(s1S";`	Seleziona i caratteri corsivi
`LPRINT CHR$(27); "(snB";`	Spessore linea (da -3 a 3)
`LPRINT CHR$(12);`	Avanzamento pagina
`LPRINT CHR$(13);`	Ritorno a capo e avanzamento riga
`LPRINT CHR$(27); "E"; CHR$(1);`	Ripristina le impostazioni di default
`LPRINT CHR$(27); "(snV";`	Imposta l'altezza verticale di font scalabili a n punti

*La lettera l (elle) appare in grassetto

CONCLUSIONI

In questo capitolo sono stati esaminati i concetti fondamentali sulle variabili. Si è visto come impostarle, come inserire dei valori, come manipolarle per generare nuovi risultati, e come visualizzare e stampare i dati. Nel prossimo capitolo verranno presentati dei nuovi enunciati che consentono di controllare il modo in cui QBasic deve eseguire il programma, conferendo in pratica al programma un 'potere decisionale'.

CAPITOLO 5

DECISIONI E RIPETIZIONI

Non è esagerato affermare che le potenzialità di un linguaggio di programmazione dipendono dalle capacità di valutazione delle condizioni e dagli enunciati di tipo decisionale disponibili. Anche un'operazione semplice come la scelta di una voce da un menu richiede l'intervento di un enunciato decisionale. L'operazione che deve svolgere il computer dipende interamente dalla scelta effettuata dall'utente.

In questo capitolo si vedrà come impostare questi enunciati di tipo decisionale. Se qualcuno non ha familiarità con questo argomento, consideri il fatto che tutto ciò che viene inserito nel computer è memorizzato in formato binario in stringhe composte da bit (otto bit formano un byte). Finora, si è detto che i due valori possibili di un bit sono 0 e 1. Tuttavia, un bit può essere facilmente interpretato in termini di una qualsiasi coppia di contrari come Sì e No, Alto e Basso, e Vero e Falso. Le decisioni all'interno di un programma si basano sempre sulla verità o falsità di una data *proposizione*. Una proposizione è semplicemente una dichiarazione che può risultare vera o falsa, ma la cui veridicità può essere determinata. Si consideri questo esempio:

Il cielo è blu

La maggior parte delle persone direbbe immediatamente che questa proposizione è vera (ad eccezione, forse, di coloro che vivono in Groenlandia). Naturalmente, il cielo non è sempre blu, neppure in Sicilia. Le nuvole, che sono bianche, a volte coprono

il blu del cielo. Dato che il colore blu del cielo sembra dipendere dalla presenza delle nuvole, si potrebbe formulare un piccolo test per determinare il colore del cielo in un qualsiasi giorno. Si direbbe:

Se *non ci sono nuvole il cielo è blu*

Questo è un enunciato chiaramente vero. I matematici direbbero che si tratta di un *assioma*; ciò equivale a dire che questa non è soltanto una buona considerazione, ma è la legge. Ci si ponga ora la domanda:

Ci sono delle nuvole?

Per portare questa domanda in un programma, la si dovrebbe porre nei seguenti termini:

È vero che non ci sono nuvole?

In caso di risposta affermativa si può dedurre che il cielo è blu; in caso contrario, invece, il cielo non è blu. Si possono a questo punto riunire tutti questi enunciati nel sillogismo seguente:

Se è vero che non ci sono nuvole il cielo è blu.
Non ci sono nuvole.
Quindi il cielo è blu.

Ciò che è stato introdotto in questa sezione è chiamato *logica proposizionale elementare*, un 'gioco' iniziato prima di Aristotele che ha conosciuto il massimo sviluppo verso la fine del diciannovesimo secolo e agli inizi del ventesimo. La logica proposizionale ha oggi trovato diverse applicazioni pratiche soprattutto in campo informatico.

Si consideri un esempio relativo ai computer, e si ritorni al menu di cui si è parlato in precedenza. Per determinare quale operazione svolgere sulla base della scelta effettuata dall'utente, si potrebbe esprimere l'enunciato seguente:

Se l'utente ha scelto "A" il programma deve fermarsi.
Se l'utente ha scelto "B" il programma deve cuocere il pane.
Se l'utente ha scelto "C" il programma deve lavare il pavimento.
L'utente ha scelto A.
Quindi il programma si ferma.

Tutti i linguaggi di programmazione consentono di impostare enunciati di questo tipo e di determinare l'operazione da svolgere sulla base della scelta effettuata dall'utente. In questo capitolo si vedrà come impostare in QBasic questi enunciati, e come utilizzare le risposte per svolgere determinate operazioni.

Un'ultima considerazione. Si sa che i computer sono particolarmente abili nello svolgere operazioni ripetitive, come sommare periodicamente delle lunghe colonne di numeri e ordinare i dati contenuti negli archivi dopo ogni aggiornamento. È possibile sviluppare dei programmi che eseguano delle singole operazioni più volte, senza che l'utente debba intervenire. In tutti i linguaggi di programmazione, ed ovviamente anche in QBasic, sono disponibili degli enunciati di tipo decisionale che facilitano notevolmente la creazione di un programma che deve svolgere operazioni di questo tipo. Una buona parte di questo capitolo è dedicata all'esamina delle strutture e degli enunciati usati per eseguire operazioni ripetitive. Verranno ora presentate alcune informazioni relative alla logica dei computer e verranno illustrati alcuni operatori necessari per impostare enunciati decisionali.

OPERATORI LOGICI E RELAZIONALI

I programmi per computer risultano utili per la loro capacità decisionale. In particolare, un programma può determinare se due numeri sono uguali e, in caso negativo, quale dei due è maggiore. Non bisogna sottovalutare l'importanza di questo enunciato. La maggior parte delle nozioni che si apprenderanno in questo capitolo sono legate a questa capacità. Ci si potrebbe ricordare dalla scuola elementare gli esercizi che si svolgevano con i simboli > e <, come:

2 è > 6?

oppure

2+1 è > 2?

Tabella 5.1 *Gli operatori relazionali*

Simbolo	Significato
<	minore di
>	maggiore di
=	uguale a
<= (o =<)	minore di o uguale a
>= (o =>)	maggiore di o uguale a
<> o ><	diverso da

In un programma, simboli come <, = e > sono chiamati *operatori relazionali* e vengono utilizzati per impostare delle condizioni che devono restituire un risultato vero o falso. Negli esempi sopra riportati, il primo enunciato è falso, mentre il secondo è vero. La Tabella 5.1 riporta tutti gli operatori relazionali disponibili.

A titolo di esempio, si considerino i tre enunciati seguenti:

2<7
-3>=1,5
-5<>2

Il primo e il terzo enunciato sono veri, mentre il secondo è falso. È possibile utilizzare gli operatori relazionali anche con le stringhe di caratteri. La procedura di ordinamento delle stringhe è simile a quella di alfabetizzazione. Un computer determina l'ordine di due caratteri in base alla loro posizione nella *tabella ASCII* (si veda l'Appendice A). ASCII è uno standard che mette in relazione 256 caratteri e codici differenti (questi codici vengono utilizzati per controllare il computer) con i 256 valori che un byte può contenere, da 0 a 255 (da 00000000 a 11111111 in notazione binaria). Per convenzione, il carattere cui è associato il valore ASCII più basso precede l'altro (o è minore dell'altro). Per esempio, sulla base della tabella ASCII, le relazioni seguenti sono vere:

"a"<"g"
"9"<"A"
"Z"<"a"

Due stringhe vengono ordinate confrontando i caratteri che le compongono, uno alla volta, fino a quando due caratteri nella stessa posizione sono differenti. Per esempio, "pala"<"palla", "Hotel"<"disco" e "9W"<"nove". Se le stringhe hanno lunghezza differente, ma quella più corta è perfettamente uguale a una parte di quella più lunga, viene considerata minore la stringa corta. Per esempio, "tasti"<"tastiera". Se le stringhe hanno la stessa lunghezza e gli stessi caratteri nelle stesse posizioni, vengono considerate uguali.

Si può a questo punto affermare che un'espressione o una condizione consiste di due espressioni (entrambe numeriche o entrambe di tipo carattere) separate da un operatore relazionale. Una condizione è vera se i valori delle espressioni soddisfano la relazione. La Tabella 5.2 riporta alcune condizioni e i relativi risultati. Nella tabella, si presuma che alle variabili *a, b, a$* e *b$* siano stati assegnati, rispettivamente, i valori 4, 6, "salve" e "arrivederci".

Alcuni di questi esempi mostrano come delle semplici condizioni si possano rivelare molto potenti grazie all'uso delle variabili. Chiunque, infatti, può fornire la risposta all'enunciato

2>3

senza neppure pensarci. Tuttavia, è possibile sostituire il numero con una variabile, come in

b>3

per ottenere diverse valutazioni durante l'esecuzione del programma, sulla base del valore contenuto in *b*.

Tabella 5.2 *Espressioni relazionali*

Condizione	Risultato
2<3	vero
INT(2.7)>2	falso
3<=3	vero
a + 5 = b	falso
-7>a	falso
"bit">"byte"	vero
"due">"DUE"	falso
b$>a$	falso
LEN(a$)<>5	falso

Si possono anche formulare condizioni più complicate, creandole sulla base di condizioni semplici come quelle sopra riportate. In questo modo, si può impostare un enunciato decisionale che determini l'operazione da svolgere a seconda di più fattori.

A questo scopo, si possono utilizzare gli *operatori logici*, AND, OR e NOT, per collegare più condizioni in un unico enunciato. La veridicità di una condizione complessa dipende dai risultati di tutte le condizioni tra loro collegate. Per esempio, l'enunciato

"Il cielo è blu" AND "Non ci sono nuvole"

è vero solo se entrambe le condizioni sono vere. Se una delle due è falsa, ci sono ad esempio delle nuvole, l'intero enunciato deve essere considerato falso.

La Tabella 5.3 riporta alcuni enunciati che utilizzano gli operatori AND, OR e NOT. La condizione complessa *cond1* AND *cond2* è vera solo se entrambe le condizioni semplici restituiscono un risultato vero. NOT viene chiamato operatore di negazione e fornisce il valore opposto della condizione specificata. Infine, la codizione complessa *cond1* OR *cond2* è vera se una delle due condizioni semplici risulta vera.

Quando si elaborano delle condizioni complesse, è utile impostare una tabella di verità. Una tabella di questo tipo elenca tutte le possibilità del caso. Per esempio, la prima riga della Tabella 5.3 mostra le diverse possibilità per le varie combinazioni delle condizioni semplici *cond1* e *cond2*.

Tabella 5.3 *Gli operatori AND, OR e NOT*

cond1	cond2	cond1 AND cond2	cond1 OR cond2	NOT cond1
vero	vero	vero	vero	falso
vero	falso	falso	vero	falso
falso	vero	falso	vero	vero
falso	falso	falso	falso	vero

Prima di procedere, si prenda in considerazione un altro esempio. L'enunciato

Il cielo è blu OR l'erba è verde

è vero se una delle due condizioni semplici risulta vera. Se il cielo è blu, l'enunciato è vero indipendentemente dal colore dell'erba. Per contro, se il cielo è nuvoloso e l'erba è marrone, l'enunciato risulta falso.

La Tabella 5.4 riporta alcune condizioni con i relativi risultati. Come già fatto in precedenza, si assuma che alle variabili *a, b, a$* e *b$* siano stati assegnati i valori 4, 6, "salve" e "Arrivederci".

Se non è chiaro il significato degli operatori stringa MID e LEN, si veda il capitolo precedente per le informazioni del caso. Si consideri ora il quinto esempio riportato nella Tabella 5.4 e si sostituiscano le variabili con i valori corrispondenti:

```
(LEN("arrivederci")=11) AND ("salve">"Arrivederci")
```

Tabella 5.4 *Alcune condizioni complesse*

Condizione	Risultato
(2<3) OR (0>1)	vero
(2<3) AND (0>1)	falso
NOT (0>1)	vero
(MID$(a$,2,1)<"Z") OR (a<>4)	falso
(LEN(b$)=11) AND (a$>B$)	vero
NOT (a$>="salve")	falso

L'operatore stringa LEN determina la lunghezza di una stringa e restituisce un valore intero. Dato che la stringa "Arrivederci" è lunga 11 caratteri, LEN fornisce il valore 11. Si sostituisca la funzione LEN con questo valore:

```
(7=7) AND ("salve">"Arrivederci")
```

La prima condizione è chiaramente vera, e porta all'enunciato

```
VERO AND ("salve">"Arrivederci")
```

Per valutare la seconda condizione, ci si ricordi che due stringhe vengono confrontate un carattere alla volta, partendo dal primo. Dalla tabella ASCII usata per determinare l'ordine dei caratteri si può rilevare che la "s" è maggiore della "A". Quindi, anche la seconda condizione è vera e porta all'enunciato

```
VERO AND VERO
```

che è naturalmente vero.

La Tabella 5.5 riporta altri operatori logici disponibili in QBasic. L'operatore XOR è conosciuto come OR *esclusivo*, a differenza dell'altro operatore OR che viene spesso chiamato OR *inclusivo*). La condizione complessa *cond1* XOR *cond2* è verrà se una delle due condizioni semplici è vera, *ma non entrambe*. Questo è il significato di esclusivo: viene esclusa la possibilità che entrambe le condizioni siano vere. Si consideri l'enunciato

Il cielo è blu XOR il cielo è nuvoloso

Questo è un esempio di OR esclusivo. Se entrambe le condizioni sono vere, l'intero enunciato deve essere falso poiché non è possibile che il cielo sia nuvoloso e blu nello stesso tempo.

Tabella 5.5 *Gli operatori XOR, EQV e IMP*

cond1	cond2	cond1 XOR cond2	cond1 EQV cond2	cond1 IMP cond2
vero	vero	falso	vero	vero
vero	falso	vero	falso	falso
falso	vero	vero	falso	vero
falso	falso	falso	vero	falso

Gli operatori EQV e IMP possono essere letti, rispettivamente, come "equivale a" e "implica". Due condizioni sono equivalenti se restituiscono lo stesso risultato. Per esempio,

Il cielo è blu EQV il cielo non è nuvoloso

Si dice che una condizione ne *implica* un'altra se la seconda condizione è vera ogni volta che è vera la prima. Per esempio,

L'erba è verde IMP l'erba non è marrone

La Tabella 5.6 riporta alcuni esempi con questi operatori.

Tabella 5.6 *Alcuni esempi con XOR, EQV e IMP*

Condizione	Risultato
(1<2) XOR (3<4)	falso
(1=2) EQV ("A">"B")	vero
(1<2) IMP (1=2)	falso

Per rendere più chiaro il significato di questi operatori, si 'traducano' in italiano gli esempi della Tabella 5.6.

La prima riga afferma che "uno può essere minore di due" oppure "tre può essere minore di quattro", ma non entrambi (OR esclusivo). Dato che le due condizioni sono entrambe vere, l'intero enunciato deve essere considerato falso.

Il secondo esempio afferma che "l'enunciato 'uno uguale a 2' equivale, in termini di veridicità, all'enunciato 'la lettera A è maggiore della lettera B'". Dato che uno è diverso da due e A non è maggiore di B, l'enunciato è nel suo insieme vero, poiché entrambe le condizioni restituiscono lo stesso risultato, falso.

Il terzo esempio è un po' più delicato. In italiano, l'enunciato può essere espresso come "uno minore di due" implica che "uno è uguale a due" (in alcuni casi, l'enunciato potrebbe prendere la forma di una struttura di tipo "Se...allora", come in "Se 1<2 allora 1=2"). Chiaramente, 1<2 non implica affatto che uno è uguale a due. Dato che la prima condizione è vera e la seconda è falsa, l'enunciato risulta falso. L'operatore IMP opera nel modo seguente: se la prima condizione è vera, la seconda deve essere vera perché l'intero enunciato risulti vero. Se la prima condizione è falsa, invece, è possibile qualunque risultato, come:

Se i maiali potessero volare, al mattino ci sarebbero i maiali sugli alberi

Le condizioni complesse possono richiedere l'uso di combinazioni di operatori aritmetici, relazionali e logici. In questo libro si fa un uso intensivo di parentesi al fine di evitare possibili ambiguità. Tuttavia, in assenza di parentesi, tutte le versioni del BASIC utilizzano lo stesso *ordine di priorità* per gli operatori. Questa gerarchia determina quali operazioni devono essere svolte per prime e in quale ordine. Le espressioni aritmetiche vengono calcolate per prime seguendo le regole presentate nel Capitolo 4; vengono quindi valutati gli operatori relazionali in modo da ottenere un risultato vero o falso. Infine, vengono presi in considerazioni gli operatori logici nell'ordine seguente: NOT, AND, OR, XOR, EQV e IMP. Ad esempio,

```
NOT 2 + 3 < 6 AND "A" < "B" OR 4 * 5 < 23
```

viene elaborata come

```
((NOT((2 + 3)<6)) AND ("A">"B")) OR (((4 * 5)+2)<23)
```

Calcolando prima le espressioni aritmetiche, si ottiene:

```
((NOT (5 < 6)) AND ("A" < "B")) OR (22 < 23)
```

quindi, valutando gli operatori relazionale, si ottiene:

```
((NOT (VERO)) AND (VERO) OR (VERO)
```

si calcola quindi NOT:

```
((FALSO) AND (VERO)) OR (VERO)
```

quindi AND:

```
(FALSO) OR (VERO)
```

e infine OR che restituisce il risultato finale:

```
VERO
```

Quindi, l'enunciato sopra riportato è, nel suo insieme, vero. Come esercizio finale, si potrebbe convertire l'enunciato in lingua italiana:

Per ottenere un risultato vero, si deve verificare almeno una di queste due condizioni: o l'opposto del risultato generato dalla condizione "due più tre minore di sei" e l'enunciato "la lettera A e minore della lettera B" sono entrambi veri, oppure cinque moltiplicato per quattro e sommato a due è minore di 23.

Whew! Per fortuna è il computer che si occupa di queste operazioni.

RICAPITOLANDO

La capacità decisionale di un programma è di importanza cruciale per la risoluzione dei problemi. I 12 operatori relazionali e logici presentati in questa sezione sono sufficienti per esprimere qualsiasi condizione necessaria per prendere una decisione.

Ora che si è visto come determinare se una condizione è vera o falsa, nella prossima sezione si imparerà ad utilizzare praticamente questa capacità.

STRUTTURE DI DECISIONE

Si consideri l'esempio presentato nell'ultimo capitolo in cui il programma, dopo una scelta effettuata da menu, doveva decidere quale operazione eseguire. Su costruzioni di questo tipo si basano le strutture di tipo decisionale dei programmi. Il formato generale è

```
IF condizione THEN enunciato
```

Se la *condizione* è vera viene eseguito l'*enunciato* che segue la parola chiave THEN. In caso contrario, il programma procede con l'istruzione nella riga successiva. Utilizzando i comandi già esaminati, si potrebbe scrivere, ad esempio:

```
IF 3>2 THEN PRINT "Sì"
```

Se questo enunciato venisse inserito in un programma, QBasic visualizzerebbe sullo schermo la stringa "Sì", dato che il numero tre è maggiore del numero due.

Si supponga di voler impartire un'altra istruzione se la condizione restituisce il valore falso. A questo scopo, si può utilizzare la parola chiave ELSE. Ad esempio, si consideri l'enunciato:

```
IF 3<2 THEN PRINT "Sì" ELSE PRINT "No"
```

Se si inserisse questo enunciato in un programma, QBasic visualizzerebbe la parola No, dato che la condizione 3<2 è falsa. Questo tipo di enunciato decisionale può essere espresso nel seguente formato generale:

```
IF condizione THEN enunciato1 ELSE enunciato2
```

La maggior parte delle versione del BASIC, incluso QBasic, supporta questo enunciato IF...THEN...ELSE inserito in una singola riga. Il Programma 5.1 fornisce un esempio di questa costruzione.

IF BLOCCO

La costruzione IF *condizione* THEN *enunciato1* ELSE *enunciato2* può contenere diversi enunciati. A questo scopo è necessario separare i singoli enunciati con il simbolo due punti (:). Per esempio:

```
IF 3>2 THEN PRINT "Sì": PRINT "Corretto"   ELSE PRINT "No"
```

Programma 5.1 *Costruzione IF...THEN...ELSE su riga singola*

```
REM Costruzione IF...THEN...ELSE su riga singola [5-1]
a$ = "Puoi votare"
INPUT "Inserisci la tua età: ", age
IF age >= 18 THEN PRINT a$ ELSE PRINT a$ + " fra"; 18 - age; "anni"
END

[Esecuzione]
Inserisci la tua età: 32
Puoi votare

[Esecuzione]
Inserisci la tua età: 15
Puoi votare fra 3 anni
```

Come si può vedere, un'istruzione di questo tipo potrebbe essere troppo lunga per essere contenuta in una sola riga della finestra di visualizzazione, o potrebbe risultare di difficile interpretazione. Un modo migliore per gestire istruzioni di questo tipo è quello di utilizzare una costruzione IF blocco.

Il formato generale è

```
IF condizione THEN
   enunciati
ELSE
   enunciati
END IF
```

Ognuno dei due blocchi di 'azioni' può contenere un qualsiasi numero di enunciati, e può addirittura includere altri blocchi IF (in questi casi si parla di IF *nidificati*; la nidificazione è un concetto molto importante nella programmazione). Si noti che in un blocco IF non si può inserire un enunciato subito dopo la parola chiave THEN.

Il Programma 5.2 richiede l'inserimento di un anno e determina se questo è bisestile. Ogni anno divisibile per quattro è un anno bisestile, ad eccezione degli anni che terminano con 00, non sono multipli di 400 e neppure divisibili per 400. Ci si ricordi che l'operatore MOD fornisce il resto di una divisione.

IL COMANDO ELSEIF

Come mostrato dal Programma 5.2, i blocchi IF possono essere usati per selezionare un'opzione tra alcune disponibili. Si può rendere più semplice questa operazione tramite un'altra parola chiave, ELSEIF. Un blocco nel formato generale

```
IF cond1 THEN
       enunciato(i)
   ELSEIF cond2 THEN
       enunciato(i)
   ELSEIF cond3 THEN
       enunciato(i)
   ELSE
       enunciato(i)
END IF
```

esamina ciascuna condizione ed esegue l'enunciato(i) che segue la *prima* condizione vera. Facendo riferimento all'esempio generale sopra riportato, se *cond1* fosse falsa, ma *cond2* e *cond3* fossero entrambe vere, verrebbero eseguiti solo gli enunciati che seguono *cond2*. Gli enunciati associati a *cond3* e alla parola chiave ELSE verrebbero ignorati.

Se non si verifica nessuna condizione, vengono eseguiti gli enunciati che seguono l'istruzione ELSE. Il blocco IF può contenere un numero illimitato di comandi ELSEIF. L'istruzione ELSE alla fine del blocco IF è opzionale. Se tutte le condizioni risultano false e non è stato inserito un comando ELSE, il blocco IF non svolge nessuna operazione.

Programma 5.2 *Uso dei blocchi IF...THEN...ELSE*

```
REM Uso dei blocchi IF...THEN...ELSE [5-2]
INPUT "Anno (xxxx): ", year
IF year MOD 4 <> 0 THEN
    PRINT year; "non è un anno bisestile."
  ELSE
    IF (year MOD 100 = 0) AND (year MOD 400 <> 0) THEN
        PRINT year; "non è un anno bisestile."
      ELSE
        PRINT year; "è un anno bisestile."
    END IF
END IF
END

[Esecuzione]
Anno (xxxx): 1800
 1800 non è un anno bisestile.
```

Il Programma 5.3 utilizza l'enunciato ELSEIF per migliorare la leggibilità del Programma 5.2 che determina gli anni bisestili. Dopo la prima esecuzione, la condizione dopo l'istruzione IF è risultata falsa, mentre la condizione dopo ELSEIF ha restituito il valore vero. Dopo la seconda esecuzione, entrambe le condizioni hanno generato il risultato falso ed è stata quindi eseguita l'istruzione dopo il comando ELSE.

Programma 5.3 *Il comando ELSEIF*

```
REM Il comando ELSEIF [5-3]
INPUT "Anno (xxxx): ", year
IF year MOD 4 <> 0 THEN
    PRINT year; "non è un anno bisestile."
  ELSEIF (year MOD 100 = 0) AND (year MOD 400 <> 0) THEN
    PRINT year; "non è un anno bisestile."
  ELSE
    PRINT year; "è un anno bisestile."
END IF
END

[Esecuzione]
Anno (xxxx): 1800
 1800 non è un anno bisestile.

[Esecuzione]
Anno (xxxx): 1600
 1600 è un anno bisestile.
```

Il Programma 5.4 utilizza la parola chiave ELSEIF per calcolare le tasse che una società deve dedurre dallo stipendio di un impiegato.

EVITARE AMBIGUITÀ

Nel BASIC standard risulta molto difficile interpretare gli enunciati che contengono molteplici comandi IF, THEN e ELSE. Si consideri

```
IF condizione1 THEN IF condizione2 THEN azione1 ELSE azione2
```

In questo esempio è difficile capire se il comando ELSE si riferisce al primo o al secondo IF. La regola generale è che il primo ELSE è sempre associato al comando IF più vicino, ed ogni ELSE successivo appartiene al comando IF più vicino cui non è ancora stato assegnato un ELSE.

In QBasic, il blocco IF non solo consente di determinare la corretta associazione degli enunciati IF e ELSE, ma permette anche di associare un comando ELSE a un qualsiasi

IF precedente. La Figura 5.1 mostra due differenti interpretazioni dell'enunciato appena presentato, utilizzano dei blocchi IF. Nella figura, il blocco a sinistra associa il comando ELSE al primo IF, mentre il blocco a destra lo associa al secondo IF.

Programma 5.4 *Calcolo delle tasse con ELSEIF*

```
REM Calcolo delle tasse con ELSEIF [5-4]
INPUT "Reddito corrente: ", pay
INPUT "Reddito dell'anno precedente: ", prior
LET ficaTax = 0
IF prior + pay < 53400 THEN
    LET ficaTax = .062 * pay
  ELSEIF prior < 53400 THEN
    LET ficaTax = .062 * (53400 - prior)
END IF
IF prior + pay < 125000 THEN
    LET ficaTax = ficaTax + .0145 * pay
  ELSEIF prior < 125000 THEN
    LET ficaTax = ficaTax + .0145 * (125000 - prior)
END IF
IF ficaTax = 0 THEN
    PRINT "Nessuna tassa."
  ELSE
    PRINT USING "Tasse =$$#,###.##"; ficaTax
END IF
END

[Esecuzione]
Reddito corrente: 2000
Reddito dell'anno precedente: 60000
Tasse =    $29.00
```

```
IF condizione1 THEN                IF condizione1 THEN
   IF condizione 2                   IF condizione2 THEN
THEN azione1                            azione1
 ELSE                                 ELSE
   azione2                              azione2
END IF                               END IF
                                   END IF
```

Figura 5.1 *Blocchi IF...THEN nidificati*

STRUTTURA SELECT CASE

Spesso, l'operazione da svolgere dipende dal valore numerico o di tipo carattere di un'espressione. Si ritorni nuovamente all'esempio del menu continuamente menzionato.

In precedenza, questa situazione è stata risolta impostando un enunciato IF in una sola riga. Benché sia possibile costruire un enunciato logico per esaminare il valore del tasto premuto dall'utente, e quindi utilizzare un blocco IF per ciascuna serie di azioni, la struttura SELECT CASE è senz'altro più idonea.

Si consideri il Programma 5.5 che è stato scritto in BASIC standard (si notino i numeri di riga). Dopo aver riscritto lo stesso programma con la struttura SELECT CASE, il Programma 5.6, questo risulta non solo di più facile interpretazione, ma anche molto più flessibile.

Un blocco SELECT CASE inizia sempre con un enunciato nella forma

```
SELECT CASE espressione
```

dove l'*espressione* genera un valore numerico o stringa. Il blocco termina con l'enunciato

```
END SELECT
```

possibilmente preceduto dall'enunciato

```
CASE ELSE
```

Programma 5.5 *Un programma in BASIC standard*

```
10 REM Un programma in BASIC standard [5-5]
20 a = 7: b = 8
30 INPUT "Inserisci un numero da 1 a 10: ", n
40 IF (n = 1) OR (n = 2) THEN PRINT "Pulisci le scarpe.": GOTO 90
50 IF (3 <= n) AND (n <= 4) THEN PRINT "Chiudi la porta.": GOTO 90
60 IF n <= 6 THEN PRINT "Spegni il gas.": GOTO 90
70 IF n = a OR n = b THEN PRINT "Una buona scelta.": GOTO 90
80 PRINT "Riprova."
90 END

[Esecuzione]
Inserisci un numero da 1 a 10: 5
Spegni il gas.
Ok
```

Gli enunciati CASE all'interno del blocco prevedono una o più possibilità per il valore dell'espressione, con le diverse possibilità separate da virgole. Ogni possibilità può essere espressa come:

una *costante*, come 1 o "Sì";
una *variabile*, come *i%* o *risultato*;
un'*espressione*, come *risultato*+1;
un *operatore relazionale* preceduto da IS e seguito da una costante, una variabile o un espressione (per esempio, IS>5);
un *intervallo* espresso nella forma X TO Y, dove X e Y sono costanti, variabili o espressioni (ad esempio, 1 TO 3).

Programma 5.6 *Uso della struttura SELECT CASE*

```
REM Uso della struttura SELECT CASE [5-6]
a = 7: b = 8
INPUT "Inserisci un numero da 1 a 10: ", n
SELECT CASE n
 CASE 1, 2
    PRINT "Pulisci le scarpe."
  CASE 3 TO 4
    PRINT "Chiudi la porta."
  CASE IS <= 6
    PRINT "Spegni il gas."
  CASE a, b
    PRINT "Una buona scelta."
  CASE ELSE
    PRINT "Riprova."
END SELECT
END
```

Quando QBasic incontra un blocco SELECT CASE, valuta l'espressione e cerca quindi il primo enunciato CASE che include il valore dell'espressione oppure, se la ricerca non ha successo, l'enunciato CASE ELSE. Se QBasic trova uno di questi enunciati, esegue le istruzioni ad esso associate. Dopo l'esecuzione, oppure se non trova nessun enunciato CASE che soddisfa l'espressione, procede con le istruzioni che seguono il blocco SELECT CASE.

Il Programma 5.7 contiene alcuni enunciati decisionali e dei blocchi nidificati. Dopo aver inserito un numero compreso tra 1 e 12 per indicare il mese dell'anno, viene determinato il numero di giorni di quel mese. Viene richiesto anche l'anno per poter gestire il mese di febbraio negli anni bisestili. Vale la pena di ricordare che se si utilizza una virgola al posto di un punto e virgola in un enunciato INPUT, la richiesta viene visualizzata senza il punto di domanda alla fine. Quando si usa un punto e virgola, invece, QBasic aggiunge il punto di domanda come accade nel BASIC standard.

Programma 5.7 *Enunciati decisionali nidificati*

```
REM Enunciati decisionali nidificati [5-7]
INPUT "Numero del mese (Gen = 1, Feb = 2, ecc.): ", month
SELECT CASE month
  CASE 9, 4, 6, 11
    numberOfDays = 30
  CASE 2
    REM Determina se si tratta di un anno bisestile
    INPUT "Anno (xxxx): ", year
    IF year MOD 4 <> 0 THEN
        numberOfDays = 28
      ELSEIF (year MOD 100 = 0) AND (year MOD 400 <> 0) THEN
        numberOfDays = 28
      ELSE
        numberOfDays = 29
    END IF
  CASE ELSE
    numberOfDays = 31
END SELECT
PRINT "Il numero di giorni del mese è "; numberOfDays
END

[Esecuzione]
Numero del mese (Gen = 1, Feb = 2, ecc.): 2
Anno (xxxx): 2000
Il numero di giorni del mese è 29
```

Programma 5.8 *Un blocco SELECT CASE con un'espressione stringa*

```
REM Un blocco SELECT CASE con un'espressione stringa [5-8]
INPUT "Inserire una parola: ", word$
SELECT CASE LEFT$(word$, 1)
  CASE "a", "e", "i", "o", "u", "A", "E", "I", "O", "U"
    pigword$ = word$ + "way"
  CASE "a" TO "z"
    pigword$ = MID$(word$, 2, LEN(word$)) + LEFT$(word$, 1) + "ay"
  CASE "A" TO "Z"
    pigword$ = UCASE$(MID$(word$, 2, 1)) + MID$(word$, 3, LEN(word$))
    pigword$ = pigword$ + LCASE$(LEFT$(word$, 1)) + "ay"
END SELECT
PRINT "La parola risultante è "; pigword$
END

[Esecuzione]
Inserire una parola: Computer
La parola risultante è Omputercay
```

Il Programma 5.8 cambia la posizione di alcune lettere che compongono una parola specificata dall'utente secondo alcuni criteri definiti tramite una struttura CASE. Se la parola da 'mescolare' inizia con una lettera maiuscola, anche la parola risultante inizierà con una maiuscola e la lettera maiuscola spostata in fondo alla parola verrà convertita in minuscolo.

CICLI

Le strutture discusse finora sono ideali per le operazioni che possono essere completate in un solo passo. Alcune volte, tuttavia, si potrebbe voler ripetere un'operazione, o un gruppo di azioni, per un dato numero di volte. Si potrebbe riscrivere lo stesso blocco di istruzioni per il numero di volte desiderato, ma questo non è il modo migliore perché risulta tedioso e poco efficiente. Inoltre, non sarebbe possibile ripetere un'operazione per un numero variabile di volte. Tutti i linguaggi di programmazione consentono di gestire eventualità di questo tipo tramite l'uso di cicli, conosciuti anche come loop.

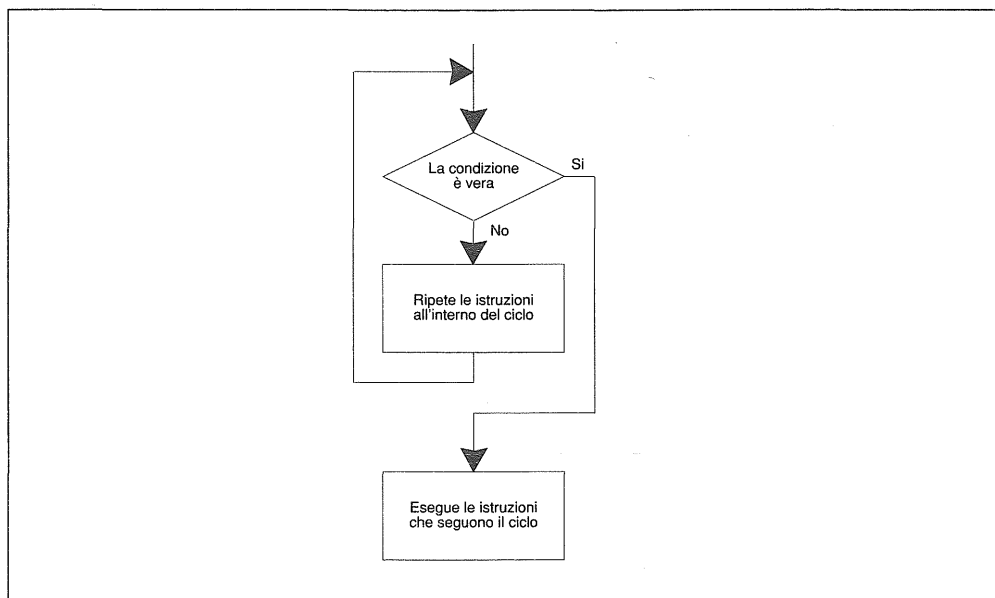



Figura 5.2 *Un diagramma di flusso relativo al ciclo DO UNTIL in cui la condizione viene verificata all'inizio*

Un ciclo ripete una sequenza di istruzioni fino a quando non si verificano determinate condizioni. La condizione può essere verificata prima o dopo la serie di comandi da eseguire. Il ciclo standard in QBasic è l'enunciato DO, che si presenta sotto forme differenti.

Quando QBasic incontra il blocco

```
DO UNTIL condizione
   enunciato(i)
LOOP
```

verifica innanzi tutto se la condizione è vera o falsa. Se risulta vera, gli enunciati all'interno del blocco vengono ignorati e il programma continua con le istruzioni alla fine del blocco (cioè dopo l'enunciato LOOP che ne contrassegna la fine). Se la condizione è falsa, vengono eseguiti tutti i comandi del blocco e l'intera procedura viene ripetuta. Il diagramma di flusso in Figura 5.2 descrive la procedura.

Il Programma 5.9 calcola la media di una lista di numeri forniti dall'utente. Il programma utilizza un ciclo per continuare a richiedere i numeri fino a quando l'utente non segnala la fine della lista tramite l'inserimento del valore -1.

Anche il blocco

```
DO
   enunciati
LOOP UNTIL condizione
```

esegue tutta la sequenza di enunciati fino a quando non si verifica la condizione specificata; tuttavia, in questo caso la condizione viene verificata dopo l'esecuzione dei comandi. Quindi, le istruzioni contenute nel blocco vengono sempre eseguite almeno una volta. Il diagramma di flusso in Figura 5.3 descrive questa procedura.

Segue un esempio relativo all'uso dei cicli. Si supponga di aver inserito in ordine alfabetico crescente 100 nomi di città nell'array città$(), e che le popolazioni corrispondenti siano contenute nella variabile POP(). Il programma 5.10 utilizza una ricerca *binaria* per trovare la popolazione di una determinata città. La ricerca viene chiamata *binaria* perché il programma determina, ad ogni passo, se i dati da cercare si trovano nella prima o nella seconda metà della lista. In questo caso, dopo ogni passo del ciclo l'intervallo di variabili viene dimezzato.

Invece di ripetere una sequenza di operazioni finché si verifica una determinata condizione, i cicli DO possono svolgere operazioni ripetitive mentre una condizione è vera. La sintassi può essere una delle seguenti:

```
DO WHILE condizione
   enunciati
LOOP
```

```
DO
   enunciati
LOOP WHILE condizione
```

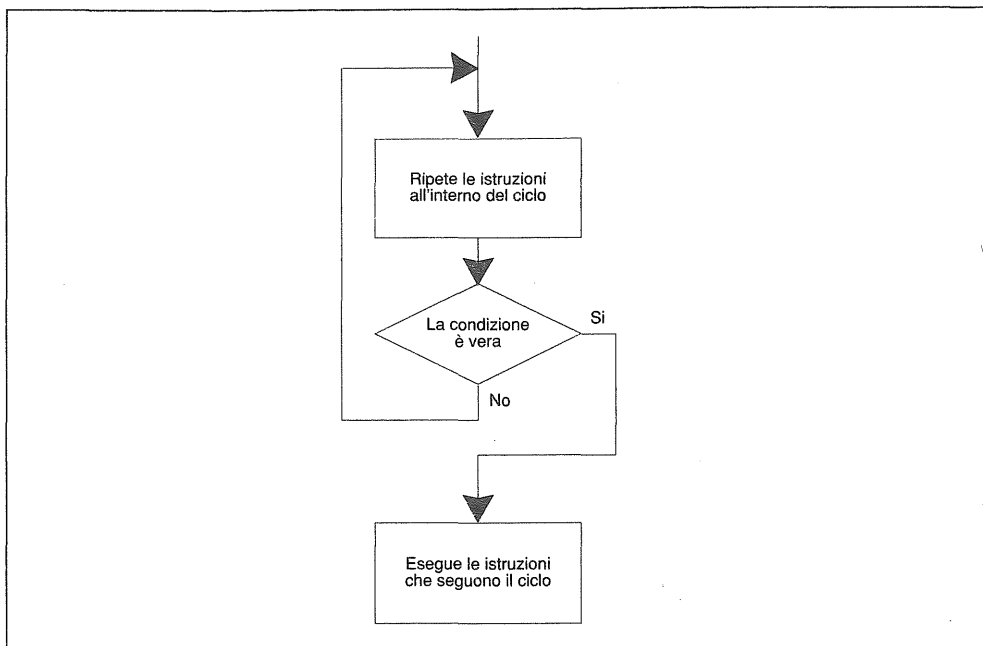



Figura 5.3 *Un diagramma di flusso relativo al ciclo DO UNTIL in cui la condizione viene verificata alla fine*

Programma 5.9 *Uso di DO UNTIL*

```
REM Calcolo della media di un gruppo di numeri [5-9]
numberOfItems = 0
sum = 0
PRINT "Inserire -1 per concludere la lista."
INPUT "Inserire un numero: ", number
DO UNTIL number = -1
  numberOfItems = numberOfItems + 1
  sum = sum + number
  INPUT "Inserire un numero: ", number
LOOP
IF numberOfItems > 0 THEN PRINT "La media è"; sum / numberOfItems
END

[Esecuzione]
Inserire -1 per concludere la lista.
Inserire un numero: 89
Inserire un numero: 94
Inserire un numero: 87
Inserire un numero: -1
La media è 90
```

```
DO WHILE numero <> -1
   'enunciati
LOOP
DO
'enunciati
LOOP WHILE (città$(middle) <> città$) AND (low <= high)
```

Figura 5.4 *Cicli DO WHILE*

Qualsiasi ciclo DO scritto con la parola chiave UNTIL, può essere riscritto con WHILE, sostituendo la condizione con la sua negazione. Per esempio, i cicli nei Programma 5.9 e 5.10 possono essere riscritti come mostrato in Figura 5.4.

Programma 5.10 *Una ricerca binaria*

```
REM Una ricerca binaria [5-10]
INPUT "Inserire il nome di una città: ", city$
low = 1
high = 100
DO
  middle = INT((low + high) / 2)
  SELECT CASE cities$(middle)
    CASE IS < city$
      low = middle + 1
    CASE IS > city$
      high = middle - 1
    CASE city$
      PRINT city$; " Ha una popolazione di"; pop(middle)
  END SELECT
LOOP UNTIL (cities$(middle) = city$) OR (low > high)
IF cities$(middle) <> city$ THEN PRINT "Città non trovata."
```

I cicli DO possono anche utilizzare due condizioni: una verificata all'inizio del ciclo e l'altra alla fine. Tuttavia, una costruzione di questo tipo può essere difficile da interpretare. Per esempio, le istruzioni riportate a sinistra in Figura 5.5 possono essere sostituite con quelle riportate a destra.

```
DO UNTIL condizione1          IF NOT condizione 1
   enunciati                     DO
LOOP UNTIL condizione2              enunciati
                                 LOOP UNTIL condi1 or condizione2
                              END IF
```

Figura 5.5 *Migliorare la leggibilità di un ciclo DO*

```
WHILE numero <> -1
      elementi = elementi+1
      somma = somma+numero
      INPUT "Inserisci un numero: ", numero
WEND
```

Figura 5.6 *Un ciclo WHILE WEND*

Si tenga presente che il ciclo WHILE...WEND, disponibile in diverse versioni avanzate del BASIC standard, esegue le stesse operazioni del ciclo DO impostato come DO WHILE *condizione*. Per esempio, il ciclo DO nel Programma 5.9 equivale al ciclo WHILE..WEND riportato in Figura 5.6.

CICLI BASATI SUI NUMERI

I cicli considerati finora sono chiamati cicli *controllati da condizioni*. Ci sono tuttavia altri modi per controllare un ciclo. La struttura FOR...NEXT, per esempio, presente in tutte le versioni del BASIC, viene denominata ciclo *controllato da contatori*. Al contatore (che non è altro che una variabile numerica) viene assegnato un valore iniziale che viene incrementato (o decrementato) dopo ogni passo del ciclo. Gli enunciati all'interno del ciclo vengono eseguiti fino a quando il contatore supera il valore specificato. Per esempio, il ciclo

```
FOR i = a TO b STEP s
   enunciati
NEXT i
```

assegna il valore *a* alla variabile *i*, e verifica se *i* è minore o uguale a *b*, se *s* è un numero positivo, o se *i* e maggiore o uguale a *b*, se *s* è un numero negativo. Se la condizione viene soddisfatta, vengono eseguiti gli enunciati compresi nel ciclo, il valore di *i* viene sostituito da *i+s*, e viene ripetuto il controllo delle variabili. La procedura continua finché la condizione restituisce un valore falso. A questo punto, il programma esce dal ciclo e continua con le istruzioni che seguono il comando NEXT.

Ad esempio, il blocco

```
FOR i = 1 TO 10 STEP 1
   PRINT "Ciao"
NEXT i
```

indica al programma di visualizzare per dieci volte sullo schermo la parola “Ciao”.

In un ciclo FOR...NEXT, *a*, *b* e *s* possono essere espressioni numeriche. Tuttavia, l'esecuzione risulta più veloce se si usano delle variabili, e ancora più veloce se si

utilizzano delle costanti. Ciò dipende dal fatto che le espressioni numeriche devono essere calcolate, e che il valore di una variabile deve essere ricercato in memoria, a differenza delle costanti che sono conosciute. Se viene omessa la parola chiave STEP, il contatore viene incrementato di 1 dopo ogni passo.

Il Programma 5.11 utilizza un ciclo FOR...NEXT per sommare i numeri dispari compresi tra 1 e 99.

Programma 5.11 *Un ciclo FOR...NEXT*

```
REM Un ciclo FOR...NEXT per sommare i numeri da 0 a 99 [5-11]
sum = 0
FOR number% = 1 TO 99 STEP 2
  sum = sum + number%
NEXT number%
PRINT sum
END

[Esecuzione]
 2500
```

In molte versioni del BASIC standard, la variabile del contatore in un ciclo FOR...NEXT deve essere intera o a precisione singola. QBasic consente di utilizzare un qualsiasi tipo di variabile come contatore. Tuttavia, si deve prestare attenzione a evitare errori di arrotondamento quando si utilizzano delle variabili a a precisione singola o doppia. Per esempio, l'enunciato

```
FOR n# = 1 TO 2 STEP 0.1
```

dovrebbe essere sostituito da

```
FOR n# = 1 TO 2.005 STEP 0.1
```

dato che la somma di 0,01 per cento volte genera un numero leggermente più grande di 1. Questo è un altro effetto sgradito del modo in cui QBasic memorizza internamente i numeri in virgola mobile. La rappresentazione binaria interna non equivale precisamente a 0,01.

Il tempo richiesto per l'esecuzione di un ciclo FOR...NEXT dipende dal tipo numerico della variabile usata come contatore. L'esecuzione risulta più rapida con le variabili intere, e rallenta progressivamente con le variabili intere lunghe, a precisione singola e a precisione doppia. La differenza in velocità tra i cicli che utilizzano variabili intere e quelli che si servono di variabili a precisione doppia è considerevole. Si dovrebbe cercare di utilizzare sempre le variabili intere.

CICLI INFINITI

La condizione che consente al ciclo di terminare è chiamata *condizione finale* e coinvolge generalmente una variabile il cui valore viene modificato dagli enunciati all'interno del ciclo. Tuttavia, se non esiste una condizione finale, o se la variabile di questa condizione non può essere modificata dalle istruzioni del ciclo, ci si trova davanti a un ciclo infinito. La Figura 5.7 riporta alcuni cicli infiniti.

Qualsiasi programma QBasic, inclusi quelli contenenti dei cicli infiniti, possono essere interrotti tramite la pressione della combinazione di tasti Ctrl-Break. QBasic ritorna alla finestra di visualizzazione ed evidenzia la riga in cui è stata interrotta l'esecuzione.

Molti programmi usano intenzionalmente dei cicli infiniti per ripetere una determinata operazione fino a quando l'utente non preme Ctrl-Break. Questa operazione potrebbe essere controllata in modo migliore tramite l'uso di una *struttura decisionale* che ripete un ciclo di istruzione fino a quando non viene premuto un determinato tasto.

Per esempio, il Programma 5.12 dovrebbe essere sostituito dal Programma 5.13 (il comando PLAY *lettera* riproduce la nota musicale corrispondente alla lettera).

Programma 5.12 *Un programma con un ciclo infinito*

```
REM Un programma con un ciclo infinito [5-12]
CLS
PRINT "Premere un tasto da A a G"
DO
  a$ = INPUT$(1)
  IF UCASE$(a$) >= "A" AND UCASE$(a$) <= "G" THEN PLAY a$
LOOP
END
```

```
DO
     PRINT "Ciao";
LOOP

sum = 0
FOR i = 1 TO 5 STEP 0
     sum = sum + 1
NEXT i

INPUT "Inserisci un numero: ", n
DO UNTIL n * n < 0
```

Figura 5.7 *Cicli infiniti*

Programma 5.13 *Il Programma 5.12 migliorato*

```
REM Il Programma 5.12 migliorato [5-13]
CLS
PRINT "Premere un tasto da A a G"
PRINT "Premere Q per uscire dal programma"
DO
  a$ = INPUT$(1)
  IF UCASE$(a$) >= "A" AND UCASE$(a$) <= "G" THEN PLAY a$
LOOP UNTIL UCASE$(a$) = "Q"
END
```

USCITA DAI CICLI E DALLA STRUTTURE DECISIONALI

I cicli hanno dei punti di entrata e uscita ben definiti. Tuttavia, alcune volte si potrebbe voler inserire un'opzione per poter uscire da un ciclo durante lo svolgimento delle operazioni. A questo scopo, si può utilizzare la parola chiave EXIT. Gli enunciati EXIT DO e EXIT FOR consentono di uscire, rispettivamente, dai cicli DO...LOOP e FOR...NEXT e di far proseguire l'esecuzione del programma con le istruzioni che seguono i cicli stessi.

```
DO
   enunciati
   IF condizione THEN EXIT DO
   enunciati2
LOOP
```

Nell'esempio sopra riportato, il blocco *enunciati1* viene sempre eseguito. Tuttavia, se la *condizione* è vera, viene eseguita l'istruzione EXIT e il programma esce dal ciclo. Se *condizione* è falsa, viene eseguito il blocco di istruzioni *enunciati2* e l'intero ciclo viene nuovamente eseguito. Il Programma 5.14 esegua una ricerca sequenziale di 100 dati nell'array non ordinato città$(), per verificare se una determinata città si trova nell'array. Si noti che quando il programma esce da un ciclo FOR...NEXT prematuramente, il contatore non viene incrementato.

RIENTRI E NIDIFICAZIONI

Quando si utilizzano diversi enunciati IF e CASE, e soprattutto quando questi vengono *nidificati*, il programma risulta difficile da interpretare e capire. Per rendere un programma più leggibile, si dovrebbero far rientrare le strutture contenute all'interno di altri enunciati. A questo scopo, si può utilizzare il tasto TAB, o aggiungere semplicemente degli spazi, in modo che le strutture nidificate siano tra

loro allineate. Per capire questo concetto, si pensi all'indice di un libro che potrebbe essere strutturato nel modo seguente:

I. Introduzione
   A. Una breve panoramica
   B. Funzioni avanzate
II. Conclusioni
   A. Prospetto riassuntivo dei comandi di QBasic
   B. Differenze tra QBasic e il BASIC standard

Programma 5.14 *Uso dell'enunciato EXIT*

```
REM Una ricerca sequenziale [5-14]
INPUT "Inserire il nome di una città: ", city$
FOR i = 1 TO 100
  IF city$(i) = city$ THEN EXIT FOR
NEXT i
IF i = 101 THEN
    PRINT "Città non trovata."
  ELSE
    PRINT "La città è l'elemento"; i; "dell'array."
END IF
```

QBasic compila correttamente un programma anche se le strutture nidificate non vengono rientrate. Tuttavia, un'indentazione corretta migliora la leggibilità di un programma e, di conseguenza, lo stile di programmazione. In questo capitolo, i blocchi nidificati e le istruzioni all'interno dei cicli sono state sempre rientrate per rendere più chiara la struttura.

I rientri vengono utilizzati anche per mostrare che un determinato blocco è contenuto, o nidificato, in un altro. In questo modo, si possono rilevare facilmente quali istruzioni LOOP sono associate agli enunciati DO, e quali comandi NEXT si riferiscono agli enunciati FOR. L'indentazione aiuta inoltre a prevenire delle nidificazioni improprie, come mostrato nella Figura 5.8. Quando i cicli vengono nidificati correttamente, tutto il ciclo interno è contenuto in quello esterno.

Il Programma 5.15 utilizza dei cicli nidificati per ridisporre gli elementi di un array. Si notino i due cicli DO WHILE nidificati e il ciclo FOR...NEXT contenuto all'interno di entrambi.

In questo capitolo si è visto come controllare l'esecuzione degli enunciati di un programma tramite condizioni logiche e strutture decisionali. A volte, tuttavia, si potrebbe voler trasferire il controllo di un programma in un punto completamente differente. Nel prossimo capitolo si vedrà come operare in situazioni del genere, utilizzano le subroutine e i sottoprogrammi.

```
DO UNTIL condizione
   FOR i = 1 TO 3
LOOP
   NEXT i

FOR i = 1 TO 5
   FOR k = 3 TO 9
NEXT i
   NEXT k
```

Nidificato in modo errato

```
DO UNTIL condizione
   FOR i = 1 TO 3
   NEXT i
LOOP

FOR i = 1 TO 5
   FOR k = 3 TO 9
   NEXT k
NEXT i
```

Nidificato correttamente

Figura 5.8 *Cicli nidificati*

Programma 5.15 *Un programma che utilizza cicli nidificati*

```
REM Cicli nidificati [5-15]
gap = INT(n / 2)
DO WHILE gap > 0
  sorted$ = "no"
  DO WHILE sorted$ = "no"
    sorted$ = "sì"
    FOR i = 1 TO n - gap
      IF array$(i) > array$(i + gap) THEN
        SWAP array$(i), array$(i + gap)
        sorted$ = "no"
      END IF
    NEXT i
  LOOP
  gap = INT(gap / 2)
LOOP
```

CAPITOLO 6

FUNZIONI, SUBROUTINE E SOTTOPROGRAMMI

Un programmatore è un persona che risolve dei problemi con l'aiuto del computer. La procedura per la risoluzione di problemi consiste di quattro operazioni distinte:

1. definizione e comprensione del problema;
2. impostazione di un algoritmo per la risoluzione del problema;
3. stesura del programma;
4. verifica del programma

La prima operazione richiede che il programmatore sappia esattamente quali dati devono essere forniti al computer, quali risultati si devono ottenere, e quale relazione esiste tra queste due componenti. La difficoltà della terza operazione (la creazione del programma) dipende dalla qualità dell'algoritmo sviluppato nel secondo punto.

Il problema originale deve essere suddiviso in una sequenza di 'sottoproblemi' che possano essere gestiti più agevolmente. Se alcuni sottoproblemi risultano ancora troppo complessi per una facile gestione, è necessaria un'ulteriore suddivisione. Il programmatore deve ripetere questa operazione fino ad ottenere una serie di sottoproblemi maneggevoli. Questa procedura viene spesso chiamata 'programmazione gerarchica' o 'divisione e conquista di un programma'.

Dopo aver suddiviso il problema originale, il programmatore risolve i sottoproblemi uno alla volta, per poi riunirli ed ottenere così la soluzione finale. Gli esperti informatici concordano nel riconoscere in questa procedura il metodo migliore per la soluzione di problemi complessi.

Il BASIC standard mette a disposizione due strumenti per la gestione dei sottoproblemi: le subroutine e le funzioni a singola riga definite dall'utente. QBasic non solo dispone di questi due strumenti, ma utilizza anche due versioni aggiornate che ricoprono un ruolo importante nella programmazione moderna: i sottoprogrammi e le procedure di funzioni. In questo capitolo si discuteranno brevemente gli strumenti del BASIC standard, in modo da rendere più evidenti i miglioramenti apportati nelle nuove versioni.

FUNZIONI A SINGOLA RIGA

QBasic dispone di numerose funzioni predefinite. La Tabella 6.1 presenta alcune delle funzioni che sono state incontrate negli esempi di questo libro.

Benché una funzione possa richiedere alcuni valori in entrata, viene sempre generato un unico risultato. Si può determinare il tipo di output esaminando il nome della funzione. Se il nome della funzione è seguito dal simbolo $, l'output sarà una stringa; in caso contrario, l'output sarà un numero.

Oltre a queste funzioni predefinite, si possono definire delle funzioni personalizzate, denominate funzioni FN. Queste nuove funzioni definite dall'utente devono iniziare con le lettere FN e il loro nome deve essere conforme alle regole esaminate per le variabili. Come per i nomi di variabile, si dovrebbe assegnare un nome descrittivo a una funzione. Per impostare una funzione FN, si deve utilizzare la parola chiave DEF seguita dalla definizione. Seguono due esempi.

```
DEF FNNome$ (n$) = LEFT$(n$, INSTR(n$, " ") - 1)
DEF FNRaddoppio (x) = 72 / x
```

Tabella 6.1 *Alcune funzioni predefinite*

Funzione	Esempio	Input	Output
INT	INT(2.6) restituisce 2	numero	numero
CHR$	CHR$(65) restituisce "A"	numero	stringa
LEN	LEN("forse") restituisce 5	stringa	numero
MID$	MID$("forse", 3, 2) restituisce "rs"	stringa, numero, numero	stringa
INSTR	INSTR("da essere", " ") restituisce 3	stringa; stringa	numero

Programma 6.1 *Una funzione FN*

```
REM Uso di una funzione FN [6-1]
DEF FNNome$ (n$) = LEFT$(n$, INSTR(n$, " ") - 1)
INPUT "Nome della persona"; name$
PRINT "Il nome proprio è "; FNNome$(name$)
END

[Esecuzione]
Nome della persona? Mario Rossi
Il nome proprio è Mario
```

La funzione FNNome$ preleva il nome proprio dal nome completo di una persona. La funzione FNRaddoppio, invece, stima il numero di anni necessari per raddoppiare il valore di un investimento con un tasso di interesse pari a x percento.La formula utilizzata è normalmente conosciuta come la Regola del 72. Per esempio, FNRaddoppio(8) genera il risultato 9; ciò significa che se si effettua investimento al tasso di interesse dell'8%, sono richiesti circa nove anni per raddoppiarne il valore. I Programmi 6.1 e 6.2 illustrano l'uso delle due funzioni appena discusse.

Le variabili x e *n$* che appaiono nelle definizioni delle funzioni sono denominate *parametri*, ed hanno un significato solo all'interno della definizione della funzione stessa. Quando si utilizza una funzione FN in un programma, al posto dei parametri appaiono delle costanti, delle variabili o delle espressioni che sono chiamate *argomenti*.

Programma 6.2 *Un altro esempio di una funzione FN*

```
REM Stima del tempo richiesto per raddoppiare un investimento [6-2]
DEF FNRaddoppio (x) = 72 / x
INPUT "Tasso di interesse in percentuale"; p
PRINT "L'investimento si raddoppierà in circa"; FNRaddoppio(p);
      "anni."
END

[Esecuzione]
Tasso di interesse in percentuale? 8
L'investimento si raddoppierà in circa 9 anni.
```

Per esempio, nel Programma 6.2 x è il parametro e p l'argomento. Come per le funzioni predefinite, l'unica restrizione per gli argomenti è che ognuno di essi deve essere del tipo appropriato: numerico o stringa.

Per esempio, FNRaddoppio(5.5), FNRaddoppio(*tasso*) e FNRaddoppio(*netto*/7) sono usi appropriati della funzione FNRaddoppio, dato che il Programma 6.2 definisce FNRaddoppio con un singolo parametro numerico. Tuttavia, FNRaddoppio(*tasso, netto*/7) e FNRaddoppio(*tasso$*) non sono funzioni valide.

Come per le funzioni predefinite, le funzioni FN definite dall'utente generano un solo risultato che può essere di tipo numerico o stringa. Si deve aggiungere il segno $ se l'output è di tipo stringa. Inoltre, si possono utilizzare i simboli di dichiarazione alla fine del nome delle variabili per indicare un tipo numerico specifico (allo stesso modo, si può specificare anche il tipo di output).

Una funzione FN può richiedere uno o più parametri. Seguono due esempi di funzioni FN che si servono di diversi parametri:

```
FNIpotenusa (a, b) = SQR(a^2 + B^2)
FNValoreFuturo (p, n, r) = p * (1 + r) ^ n
```

La funzione FNIpotenusa fornisce la lunghezza dell'ipotenusa di un triangolo rettangolo, data la lunghezza dei lati *a* e *b*. La funzione FNValoreFuturo, invece, calcola il saldo di un conto corrente bancario dopo *n* periodi di interesse dopo aver depositato *p* lire a un tasso di interesse pari a *r* per periodo. Il Programma 6.3 utilizza la funzione ipotenusa.

Il Programma 6.4 usa la funzione valore futuro. Usando i dati riportati di seguito, il programma calcola il saldo di un conto corrente bancario dopo aver depositato 100 dollari per cinque anni all'interesse dell'8% composto trimestralmente. L'interesse matura quattro volte all'anno al tasso del 2% per periodo. Ci saranno quindi 20 periodi di interesse.

Si possono utilizzare delle espressioni logiche quando si definiscono delle funzioni che devono svolgere operazioni di tipo IF...THEN. Le espressioni logiche possono essere vere o false. Tuttavia, invece di utilizzare i valori vero e falso, il programma interpreta il valore falso come 0 e il valore vero come -1. Il Programma 6.5 illustra un'espressione logica di questo tipo. Nella quarta riga, l'espressione logica genera il valore -1 che viene utilizzato come parte dell'espressione numerica.

Programma 6.3 *Un esempio con la funzione FNIpotenusa*

```
REM Calcolo dell'ipotenusa di un triangolo rettangolo  [6-3]
DEF FNIpotenusa (a, b) = SQR(a ^ 2 + b ^ 2)
INPUT "Specificare la lunghezza dei cateti: ", leg1, leg2
PRINT "La lunghezza dell'ipotenusa è"; FNIpotenusa(leg1, leg2)
END

[Esecuzione]
Specificare la lunghezza dei cateti: 3,4
La lunghezza dell'ipotenusa è 5
```

Programma 6.4 *Un esempio con la funzione FNValoreFuturo*

```
REM Calcolo del valore futuro [6-4]
DEF FNValoreFuturo (p, n, r) = p * (1 + r) ^ n
INPUT "Ammontare del deposito"; p
INPUT "Numero di periodi"; n
INPUT "Tasso di interesse per periodo"; r
PRINT USING "Il risultato è $####.##"; FNValoreFuturo(p, n, r)
END

[Esecuzione]
Ammontare del deposito? 100
Numero di periodi? 20
Tasso di interesse per periodo? .02
Il risultato è $ 148.59
```

Il Programma 6.6 utilizza delle espressioni logiche per definire la funzione FNMax che determina il numero più grosso tra due numeri. Se x è maggiore di y, l'espressione x>=y risulta vera e genera il valore -1; in caso contrario, viene fornito il valore 0. Quindi, in questo caso, il valore di FNMax(*x*, *y*) sarà

-(x>=y)*x-(y>x)*y
=-(-1)*x-(0)*y
=x

Programma 6.5 *Valutazione di espressioni logiche*

```
REM Valutazione di espressioni logiche [6-5]
PRINT 2 = 1
PRINT 1 < 2
PRINT (1 < 2) * 3 + (4 < 5) * 6
END

[Esecuzione]
  0
 -1
 -9
```

Si può facilmente verificare che la funzione restituisce *y* nel caso che *y* sia maggiore di *x*, e *x* quando *x* e *y* sono uguali. Dato che QBasic considera il valore falso uguale a 0 e il valore vero uguale a -1, si possono definire delle funzioni numeriche i cui possibili valori possano essere interpretati come vero e falso, anche se in realtà vengono generati dei valori 0 e -1. A titolo di esempio, si supponga di voler determinare se un carattere è scritto in maiuscolo o in minuscolo. Se *a$* contiene un singolo carattere, l'enunciato

```
IF (a$ >= "A") AND (a$ <= "z") THEN azione
```

esegue *azione* quando a$ contiene una lettera in maiuscolo. Per rendere la decisione più ovvia, si potrebbe definire la funzione FNMaiuscolo con l'enunciato

```
DEF FNMaiuscolo (a$) = (a$ >= "A") AND (a$ <= "z")
```

e scrivere quindi l'enunciato IF seguente:

```
IF FNMaiuscolo (a$) THEN azione
```

Programma 6.6 *Una funzione che utilizza espressioni logiche*

```
REM Identifica il numero più grande    [6-6]
DEF FNMax (x, y) = -(x >= y) * x - (y > x) * y
INPUT "Inserire due numeri separati da una virgola: ", x, y
PRINT "Il numero più grande è"; FNMax(x, y)
END

[Esecuzione]
Inserire due numeri separati da una virgola: 3,7
Il numero più grande è 7
```

Si noti che i parametri usati per definire una funzioni non hanno *nessun significato* al di fuori della definizione stessa. Si possono cambiare i nomi dei parametri in qualsiasi altro nome dello stesso tipo senza influire sulla definizione.

Il compilatore, in pratica, riserva delle locazioni di memoria speciali per i parametri che vengono utilizzate solo quando viene impartita la funzione. I nomi che il compilatore assegna a queste locazioni non hanno nulla a che vedere con i nomi di variabile usati dal programma. Quindi, nel caso in cui una funzione FN utilizzi come parametro una variabile che appare nel programma, il valore della variabile *non cambia* nel momento in cui si esegue la funzione. Il Programma 6.7 dimostra questa caratteristica. Se si pensa che il secondo numero generato debba essere 5, si consideri che la definizione della funzione può essere vista come:

```
DEF FNTriple (valoreNumerico) = 3 * valoreNumerico
```

Alcune delle funzioni predefinite di QBasic, come TIME$ e ERR, non richiedono parametri (TIME$ fornisce l'ora corrente e ERR restituisce il codice dell'errore verificatosi più di recente). Anche le funzioni definite dall'utente possono essere prive di parametri. Per esempio, la funzione

```
DEF FNOra$ = "L'ora corrente è " + TIME$
```

può essere eseguita tramite il semplice enunciato PRINT FNOra$.

Programma 6.7 *Dimostrazione del fatto che i nomi di variabile nelle funzioni non hanno significato*

```
REM Triplo di un numero  [6-7]
x = 2
DEF FNTriple (x) = 3 * x
PRINT FNTriple(5)
PRINT x
END

[Esecuzione]
 15
 2
```

PROCEDURE DI FUNZIONI

QBasic dispone di una versione più avanzata di funzioni che non sono disponibili nel BASIC standard: la *procedura di funzione*. Le procedure di funzioni vengono definite in finestre separate, consistono di alcune righe di programma e si adattano perfettamente alla programmazione strutturata.

Il blocco di enunciati che definisce una procedura di funzione inizia con un'istruzione nella forma

```
FUNCTION NomeFunzione (lista di parametri)
```

e termina con l'enunciato END FUNCTION. I parametri in *lista* sono separati da virgole. Quando viene richiamata una funzione, il programma assegna dei valori a questi parametri e gli enunciati nel blocco utilizzano questi valori per determinare il risultato della funzione. Normalmente, l'enunciato che precede il comando END FUNCTION ha la forma

```
NomeFunzione = valore
```

e genera il risultato della funzione. D'ora in poi si utilizzerà la parola "funzione" per indicare "procedura di funzione".

Le funzioni non vengono digitate nella finestra di visualizzazione, ma in una finestra separata creata per contenere ciascuna funzione. Per definire una funzione, si proceda nel modo seguente:

1. spostare il cursore su una nuova riga;
2. se *NomeFunzione* è il nome della funzione, digitare FUNCTION *NomeFunzione* e premere Invio;

3. Una volta premuto Invio, viene aperta una nuova finestra specifica per la funzione. La riga FUNCTION *NomeFunzione* appare nella parte superiore dello schermo, seguita da una riga vuota e dalle parole END FUNCTION;
4. digitare la funzione in questa finestra come se ci si trovasse nella finestra di visualizzazione;
5. per ritornare al programma principale, premere i tasti Maiusc-F2. Si prema nuovamente questa combinazione di tasti per ritornare alla finestra della funzione. La combinazione di tasti Maiusc-F2 consente di passare alternativamente tra una finestra e l'altra.

Programma 6.8 *Una semplice funzione definita dall'utente*

```
REM Triplo di un numero  [6-8]
number = 5
PRINT Triple(number)
END

FUNCTION Triple (x)
  Triple = 3 * x
END FUNCTION
```

Negli esempi di questo libro tutte le definizioni di funzione sono state inserite dopo il comando END del programma principale, e ciascuna definizione viene separata dalle altre con una riga vuota. Il Programma 6.8 contiene una funzione elementare. Si noti che quando l'enunciato Triple = 3**x* viene usato per assegnare un valore al nome della funzione, nessun parametro segue il nome della funzione.

Il Programma 6.9 è uguale al Programma 6.6 ad eccezione del fatto che viene utilizzata una procedura per definire una funzione. Il significato risulta più chiaro nella definizione della procedura di funzione.

Programma 6.9 *Una versione procedurale della funzione del Programma 6.6*

```
REM Identifica il numero più grande   [6-9]
INPUT "Inserire due numeri separati da una virgola:", x, y
PRINT "Il numero più grande è"; Max(x, y)
END

FUNCTION Max (x, y)
  IF x >= y THEN
      Max = x
    ELSE
      Max = y
  END IF
END FUNCTION
```

Il Programma 6.10 include la definizione della funzione IsALeapYear. Per rendere più chiara la definizione della funzione, il programma assegna all'inizio della definizione i valori appropriati alle variabili numeriche denominate true e false. Si noti, inoltre, che si può assegnare un valore a un nome di funzione in più di un enunciato all'interno della definizione stessa.

Il Programma 6.11 utilizza una funzione che non può essere definita in una singola riga. La funzione determina il numero di parole contenute in una frase contando gli spazi e aggiungendo uno.

Programma 6.10 *Una funzione definita dall'utente*

```
REM Determina il numero di giorni nell'anno specificato [6-10]
INPUT "Quale anno"; year
PRINT "L'anno"; year; "ha ";
IF IsALeapYear(year) THEN
    PRINT "366 giorni."
  ELSE
    PRINT "365 giorni."
END IF
END

FUNCTION IsALeapYear (y)
  true = -1
  false = 0
  IF y MOD 4 <> 0 THEN
      IsALeapYear = false
    ELSEIF (y MOD 100 = 0) AND (y MOD 400 <> 0) THEN
      IsALeapYear = false
    ELSE
      IsALeapYear = true
  END IF
END FUNCTION

[Esecuzione]
Quale anno? 1900
L'anno 1900 ha 365 giorni.
```

VARIABILI LOCALI, STATICHE E CONDIVISE

Quando lo stesso nome di variabile appare in una procedura di funzione e nel programma principale, QBasic assegna alle variabili due identità differenti e le tratta come variabili diverse. Le variabili in una funzione sono dette *locali* rispetto al blocco in cui risiedono. Ogni volta che viene richiamata la funzione, QBasic riserva delle nuove locazioni di memoria per contenere i valori corrispondenti. Quando il programma esce dal blocco, QBasic 'dimentica' queste locazioni di memoria. In

questo modo, quando si richiama la funzione, non solo la memoria viene gestita in modo efficiente, ma le variabili numeriche locali vengono azzerate e quelle stringa convertite in stringa nulla.

Programma 6.11 *Una funzione definita dall'utente che contiene un ciclo*

```
REM Conta il numero di parole in una frase  [6-11]
INPUT "Inserire la frase: ", sentence$
PRINT "La frase contiene";
PRINT NumberOfWords(sentence$); "parole."
END

FUNCTION NumberOfWords (a$)
  FOR i = 1 TO LEN(a$)
    IF MID$(a$, i, 1) = " " THEN spaces = spaces + 1
  NEXT i
  NumberOfWords = spaces + 1
END FUNCTION

[Esecuzione]
Inserire la frase: L'abito non fa il monaco
La frase contiene 5 parole.
```

Una variabile che appare nella definizione di una procedura di funzione può anche essere dichiarata come variabile *statica.* Analogamente alle variabili locali, le variabili statiche non hanno alcuna relazione con qualsiasi altra variabile al di fuori del blocco di enunciati che definisce la funzione; tuttavia, QBasic conserva il loro valore quando ritorna al programma principale. È quindi possibile usare o modificare queste variabili ogni volta che si richiama una funzione che le contiene. L'enunciato STATIC *lista di variabili* consente di specificare le variabili da dichiarare come statiche.

Le variabili che appaiono nella definizione di una procedura di funzione e che vengono riconosciute dall'intero programma sono chiamate variabili *globali* o *condivise.* Nel blocco di enunciati che definiscono la funzione, QBasic presume che qualsiasi variabile non dichiarata come statica o globale debba essere considerata come variabile locale. Quindi, *true* e *false* nel programma 6.10 sono variabili locali.

Per rendere i programmi più facili da interpretare, QBasic permette di dichiarare delle variabili condivise tramite l'enunciato SHARED *lista di variabili.* Analogamente a quanto visto per STATIC, le dichiarazioni di tipo SHARED devono seguire l'enunciato FUNCTION e devono precedere qualsiasi altro comando presente nel blocco di definizione. Solamente i comandi REM e DEFtipo possono precedere gli enunciati di dichiarazione.

L'output dei tre programmi riportati in Figura 6.1 illustrano gli effetti dei due nuovi tipi di variabile nella definizione della funzione Triplo.

```
REM n shared
n = 2
PRINT Triplo(5);
PRINT n;
PRINT Triplo(6);
PRINT a
END

FUNCTION Triplo(x)
   SHARED a, n
   a = n
   n = 3 * x
   Triple = n
END FUNCTION

[Esecuzione]
15  15  18  15
```

```
REM n local
n = 2
PRINT Triplo(5);
PRINT n;
PRINT Triplo(6);
PRINT a
END

FUNCTION Triplo(x)
   SHARED a
   a = n
   n = 3 * x
   Triple = n
END FUNCTION

[Esecuzione]
15  2  18  0
```

```
REM n static
n = 2
PRINT Triplo(5);
PRINT n;
PRINT Triplo(6);
PRINT a
END

FUNCTION Triplo(x)
   SHARED a
   STATIC N
   a = n
   n = 3 * x
   Triple = n
END FUNCTION

[Esecuzione]
15  2  18  15
```

Figura 6.1 *Variabili statiche, locali e condivise*

DELLE BUONE RAGIONI PER USARE LE FUNZIONI

Anche se può sembrare complicato, ci sono degli ottimi motivi per cui si dovrebbero utilizzare delle funzioni. Ognuno degli obiettivi seguenti può essere facilmente raggiunto grazie alla possibilità di dichiarare variabili locali, statiche e condivise.

1. Durante la progettazione dello scheletro del programma, quando ci si rende conto che in un determinato punto è necessaria una funzione, si può inserire semplicemente il nome della funzione, continuare la progettazione, e sviluppare successivamente la funzione stessa;
2. alcune volte il medesimo algoritmo deve essere eseguito più volte in un programma. Se l'algoritmo viene specificato come funzione, si evita di dover ridigitare la stessa formula, si rende il programma più leggibile e si semplificano le operazioni di collaudo;
3. si possono usare delle funzioni di un programma in un altro programma. Come programmatore, si può creare una *libreria* di funzioni che potranno essere duplicate nei vari programmi quando necessario. Inoltre, se si utilizzano variabili locali o statiche, si evita il problema della duplicazione dei nomi di variabile.

SUBROUTINE

Alcune volte, tuttavia, le funzioni non bastano per risolvere un problema. Si supponga, ad esempio, di voler ottenere più di un risultato. In questo caso, si dovrebbe scrivere una *subroutine.*

Una subroutine è una porzione di programma che risiede al di fuori del blocco di istruzioni principale del programma, termina con l'enunciato RETURN e viene richiamata tramite il comando GOSUB. Alla prima riga di una subroutine deve essere associato un numero di riga o un'etichetta. Un numero di riga è un intero non negativo composto al massimo da 40 cifre. Un'etichetta è un nome, simile a un nome di variabile, che termina con i due punti (:). Il numero di riga o l'etichetta deve essere posto davanti alla prima riga della subroutine.

Ad esempio, Il comando del BASIC standard

```
GOSUB 1000
```

indica al programma di cercare ed eseguire l'istruzione identificata con il numero 1000. L'esecuzione procede con gli enunciati che seguono numero di riga 1000 e termina quando viene incontrato il comando

```
RETURN
```

che riporta l'esecuzione all'enunciato che segue l'istruzione GOSUB 1000.

Ogni volta che viene eseguito l'enunciato GOSUB *nomeEtichetta* o GOSUB *numeroDiRiga,* il programma *salta* alla riga specificata. Ciò significa che l'esecuzione prosegue con l'enunciato che si trova nella riga specificata e non con quello che segue il comando GOSUB. Quando QBasic incontra la parola chiave RETURN; il controllo ritorna all'istruzione posta subito dopo l'enunciato GOSUB. Le subroutine vengono normalmente inserite alla fine del programma e sono separate dal blocco principale di istruzioni tramite la parola chiave END.

Il Programma 6.12 utilizza una subroutine per apportare delle modifiche ai dati relativi a un cliente. Il programma converte inizialmente la quantità di denaro dovuto in centesimi e determina quindi il numero di banconote e monete necessario. Alla subroutine è stato assegnato il nome ReportNumerOfUNits.

Questa subroutine mostra come sia possibile suddividere un programma in porzioni più piccole e maneggevoli. Oltre a supportare le subroutine, QBasic dispone di un altro strumento, chiamato *sottoprogramma,* che opera come una subroutine, ma che mette a disposizione funzioni più avanzate. I sottoprogrammi sono considerati come il più importante miglioramento apportato da QBasic al BASIC standard.

Programma 6.12 *Uso del comando GOSUB*

```
REM Calcolo del resto  [6-12]
INPUT "Costo totale"; cost
INPUT "Somma pagata"; paid
changeDue% = 100 * (paid - cost)   'Cambio in centesimi
PRINT " Cambio:"
unit% = 100             '100 centesimi = 1 dollaro
unitName$ = "dollaro"
GOSUB ReportNumberOfUnits
unit% = 25              '25 centesimi = 1 quarto
unitName$ = "quarto"
GOSUB ReportNumberOfUnits
unit% = 10              '10 centesimi = 1 dime
unitName$ = "dime"
GOSUB ReportNumberOfUnits
unit% = 5               '5 centesimi = 1 nickel
unitName$ = "nickel"
GOSUB ReportNumberOfUnits
unit% = 1
unitName$ = "centesimo"
GOSUB ReportNumberOfUnits
END

ReportNumberOfUnits:
  number% = changeDue% \ unit%
  IF number% > 0 THEN
      IF number% > 1 AND (unitName$<>"dime" AND _
         unitName$<>"nickel") THEN unitName$ = _
         LEFT$(unitName$,(LEN(unitName$)-1)) + "i"
      PRINT number%; unitName$
  END IF
  changeDue% = changeDue% - (number% * unit%)
RETURN

[Esecuzione]
Costo totale? 1.65
Somma pagata? 5.00
 Cambio:
 3 dollari
 1 quarto
 1 dime
```

I SOTTOPROGRAMMI

Un sottoprogramma è una combinazione di una subroutine e di una funzione. Come una subroutine, un sottoprogramma viene raggiunto tramite un'apposita istruzione

ed esegue delle operazioni specifiche e, come una funzione, un sottoprogramma richiede dei parametri e consente di dichiarare i tipi di variabili che devono essere utilizzate. Ciascun sottoprogramma viene definito da un blocco che inizia con un enunciato nella forma

```
SUB NomeSottoprogramma (lista di parametri)
```

e che finisce con l'istruzione END SUB. Quando si inserisce un enunciato ce inizia con la parola chiave SUB, viene aperta una finestra specifica per i sottoprogrammi. Un sottoprogramma viene richiamato tramite l'enunciato

```
CALL NomeSottoprogramma (lista di argomenti)
```

dove il numero e il tipo in *lista di argomenti* devono corrispondere a quelli di *lista di parametri.* Analogamente alle funzioni, la lista degli argomenti può contenere costanti, variabili ed espressioni. Quando si richiama un sottoprogramma, i valori degli argomenti vengono passati ai parametri e utilizzati dagli enunciati del sottoprogramma. Dopo l'esecuzione di tutti questi enunciati, o quando viene incontrata l'istruzione EXIT SUB, il controllo ritorna al comando che segue l'enunciato CALL.

Il Programma 6.13 è simile al Programma 6.12, ma utilizza un sottoprogramma al posto della subroutine. Il nuovo programma passa dei valori al sottoprogramma, a differenza di quanto accade nel Programma 6.12 in cui gli stessi valori devono essere assegnati a delle variabili prima dell'enunciato GOSUB.

Gli enunciati CALL e GOSUB potrebbero sembrare equivalenti. Entrambi i comandi passano il controllo a una nuova sezione del programma, eseguono una determinata operazione e ritornano al punto originale. Tuttavia, la somiglianza tra questi due enunciati termina qui. Le considerazioni seguenti rivelano delle importanti differenze tra i sottoprogrammi e le subroutine. Da questo momento in avanti, nei programmi di esempio verranno utilizzati esclusivamente i sottoprogrammi.

1. Il comando GOSUB sposta semplicemente il controllo del programma in una nuova sezione; tuttavia, se non si presta attenzione nella stesura delle subroutine, QBasic potrebbe accedere accidentalmente a delle istruzioni che compongono la subroutine senza che sia stato utilizzato il comando GOSUB. Ciò può causare risultati imprevedibili. Il comando CALL, invece, richiama una sezione protetta di enunciati a cui QBasic non può accedere se non esplicitamente istruito. Non accade nulla di grave se si omette la parola chiave END alla fine del Programma 6.13, mentre si possono verificare dei problemi se la si omette nel Programma 6.12;
2. le subroutine non possono utilizzare delle variabili locali. Tutte le variabili equivalgono alle variabili condivise di un sottoprogramma. Quindi, quando si scrive una subroutine, bisogna assicurarsi di non utilizzare dei nomi già assegnati, a meno che non sia espressamente richiesto. Se si utilizza un nome

di variabile già assegnato, si potrebbero perdere dei risultati già generati da altre porzioni del programma;

3. GOSUB non consente di definire delle subroutine con delle variabili generiche (parametri). Prima di utilizzare un enunciato GOSUB, bisogna assegnare alle variabili appropriate i valori necessari per lo svolgimento di quella subroutine. Il programmatore deve sempre ricordare quali nomi sono stati utilizzati per una determinata subroutine. I sottoprogrammi, invece, consentono di definire delle variabili generiche, e di svolgere successivamente una determinata operazione usando quelle variabili, dei valori o delle espressioni;
4. tutte le variabili nelle subroutine restano in memoria per tutta la durata del programma. Quindi, viene riservata delle memoria per delle variabili temporanee che vengono utilizzate solo nelle subroutine. Le variabili locali dei sottoprogrammi, invece, riservano lo spazio di memoria necessario solo quando vengono utilizzate;
5. benché QBasic consenta di utilizzare delle etichette per identificare le subroutine, un enunciato GOSUB *etichetta* non evidenzia le variabili che vengono utilizzate dalla subroutine. Risulta quindi molto più difficile interpretare un programma, soprattutto se si devono apportare delle modifiche dopo un certo periodo di tempo. I sottoprogrammi non incorrono in questo inconveniente grazie alla loro lista di parametri.

Programma 6.13 *Uso dei sottoprogrammi*

```
REM Calcolo del resto  [6-13]
INPUT "Costo totale"; cost
INPUT "Somma pagata"; paid
changeDue% = 100 * (paid - cost)
PRINT " Change:"
CALL ReportNumberOfUnits(100, "dollaro")
CALL ReportNumberOfUnits(25, "quarto")
CALL ReportNumberOfUnits(10, "dime")
CALL ReportNumberOfUnits(5, "nickel")
CALL ReportNumberOfUnits(1, "centesimo")
END

SUB ReportNumberOfUnits (unit%, unitName$)
  SHARED changeDue%
  number% = changeDue% \ unit%
  IF number% > 0 THEN
      IF number% > 1 AND (unitName$<>"dime" AND _
         unitName$<>"nickel") THEN unitName$ = _
         LEFT$(unitName$,(LEN(unitName$)-1)) + "i"
    PRINT number%; unitName$
  END IF
  changeDue% = changeDue% - (number% * unit%)
END SUB
```

PASSAGGIO PER RIFERIMENTO E PASSAGGIO PER VALORE

I Programmi 6.14 e 6.15 assegnano dei valori alle variabili *state$* e *pop&* e non effettuano ulteriori assegnazioni a queste variabili. Tuttavia, i valori di queste variabili visualizzati alla fine del Programma 6.14 sono cambiati.

Quando l'enunciato CALL utilizza una variabile come argomento, come accade nel Programma 6.14, QBasic esegue un 'passaggio per riferimento'. QBasic fa riferimento alle variabili specificate nell'enunciato CALL al posto dei parametri che appaiono nella definizione del sottoprogramma. Ad esempio, nel Programma 6.14 l'enunciato CALL DisplayInfo(*state$, pop&*) esegue in realtà le istruzioni seguenti:

```
state$ = LEFT$(state$, 2)
pop& = pop& / 1000000
PRINT pop&; "milioni di persone vivono in "; state$
```

Per ogni argomento viene utilizzata una sola locazione di memoria. La Figura 6.2 mostra l'uso della memoria per la variabile *pop&*. Inizialmente, il programma principale riserva una locazione per memorizzare il valore di *pop&* (Figura 6.2a). Quando QBasic richiama il sottoprogramma, il parametro b& diventa il nuovo nome di variabile per il sottoprogramma (Figura 6.2b). Quando il contenuto di b& viene diviso per 1.000.000, il valore di questa locazione di memoria diventa 24 (Figura 6.2c). Al completamento del sottoprogramma, il parametro b& viene dimenticato; tuttavia, il suo valore è contenuto in *pop&* (Figura 6.2d). In questo caso, si dice che la variabile *pop&* è stata 'passata per riferimento'.

Una conseguenza immediata del passaggio per riferimento è che, all'interno del sottoprogramma, si possono assegnare dei nuovi valori agli argomenti usati nell'enunciato CALL; ciò spiega il comportamento del Programma 6.14.

Quando si usa una costante o un'espressione in un enunciato CALL, come nel Programma 6.15, QBasic esegue un 'passaggio per valore'. Ciò significa che la costante o il valore dell'espressione viene inserito in una nuova locazione di memoria creata per il parametro associato. Si otterrebbe lo stesso effetto se si assegnasse al parametro il valore della costante o dell'espressione all'inizio del sottoprogramma.

Ad esempio, nel Programma 6.15, l'enunciato CALL DisplayInfo(*state$*+"",*pop&*+0) esegue in realtà le istruzioni seguenti:

```
a$ = state$ + ""
b& = pop& + 0
a$ = LEFT$(a$, 2)
b& = b& / 1000000
PRINT b&; "milioni di persone vivono in "; a$
```

Programma 6.14 *Dimostrazione di un passaggio per riferimento*

```
REM Dimostrazione di un passaggio per riferimento [6-14]
state$ = "CALIFORNIA"
pop& = 24000000
CALL DisplayInfo(state$, pop&)
PRINT state$; pop&

SUB DisplayInfo (a$, b&)
  a$ = LEFT$(a$, 2)
  b& = b& / 1000000
  PRINT b&; "milioni di persone vivono in "; a$
END SUB

[Esecuzione]
 24 milioni di persone vivono in CA
CA 24
```

Programma 6.15 *Dimostrazione di un passaggio per valore*

```
REM Dimostrazione di un passaggio per valore [6-15]
state$ = "CALIFORNIA"
pop& = 24000000
CALL DisplayInfo(state$ + "", pop& + 0)
PRINT state$; pop&
END

SUB DisplayInfo (a$, b&)
  a$ = LEFT$(a$, 2)
  b& = b& / 1000000
  PRINT b&; "milioni di persone vivono in "; a$
END SUB

[Esecuzione]
 24 milioni di persone vivono in CA
CALIFORNIA 24000000
```

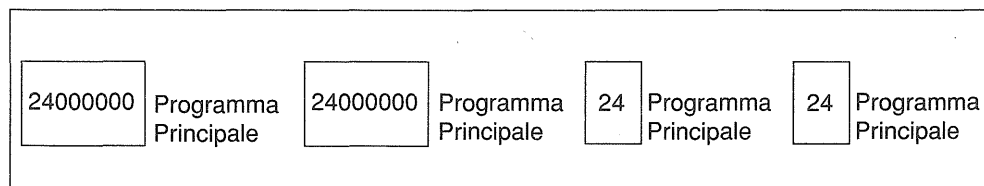



Figura 6.2 *Passaggio per riferimento di una variabile a un sottoprogramma*

Per ciascun argomento vengono utilizzate due locazioni di memoria. La Figura 6.3 mostra l'uso della memoria per la variabile *pop&* quando viene 'passata per valore'. Inizialmente, il programma principale riserva una locazione per memorizzare il valore di *pop&* (Figura 6.3a). Quando QBasic richiama il sottoprogramma, una seconda locazione di memoria temporanea per il parametro b& viene riservata per l'uso del sottoprogramma e il valore di b& viene copiato in questa locazione (Figura 6.3b). Quando il contenuto di b& viene diviso per 1.000.000, il valore di b& diventa 24 (Figura 6.3c). Al completamento del sottoprogramma, la locazione di memoria di b& 'sparisce' (Figura 6.3d) e il valore della variabile *pop&* resta lo stesso.

Da queste spiegazioni si può capire perché il programma 6.15 non cambia i valori di *state$* e *pop&*. Nel caso di un passaggio per valore, dato che i parametri possono essere sostituiti solo dai valori degli argomenti, QBasic non passa al sottoprogramma le variabili reali.

Come ulteriore esempio, si consideri il Programma 6.16 che richiama il sottoprogramma Insert con una costante come primo argomento. Il sottoprogramma visualizza la stringa Jonathan Livingston Sea Gull; tuttavia, la stringa non è disponibile per l'uso nella parte principale del programma. Se l'enunciato CALL nel Programma 6.16 venisse sostituito dai due enunciati

```
item$ = "Jonathan Sea Gull"
CALL Insert(item$, "Livingston ", 10)
```

in modo da utilizzare una variabile come primo argomento, il sottoprogramma Insert non solo visualizzerebbe la stringa Jonathan Livingston Sea Gull, ma passerebbe anche questa stringa alla variabile *item$*. Altri comandi all'interno del programma potrebbero quindi utilizzare la nuova stringa contenuta in *item$*.

A volte, si potrebbero voler memorizzare dei dati in una variabile da passare a un sottoprogramma, e non volere che il sottoprogramma modifichi il valore della variabile. Il Programma 5.15 ottiene questo risultato per le variabili *state$* e *pop&* utilizzando le espressioni *state$*+"" e *pop&*+0 come argomenti e causando quindi un passaggio per valore. QBasic mette a disposizione un'altra soluzione: se quando si richiama un sottoprogramma un argomento variabile è racchiuso tra una coppia di parentesi extra, il valore della variabile viene passato per valore al sottoprogramma.

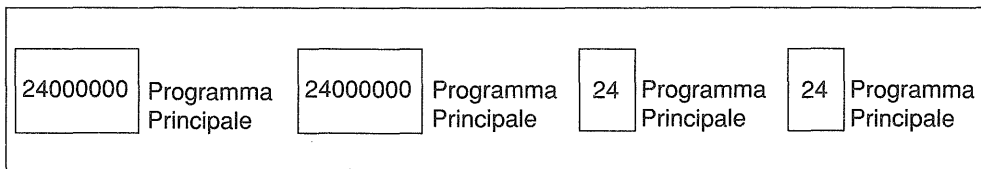



Figura 6.3 *Passaggio per riferimento di una variabile a un sottoprogramma*

Dopo l'esecuzione del sottoprogramma, la variabile conterrà il suo valore originale. Per esempio, se si sostituisse l'enunciato CALL nel Programma 6.16 con i tre enunciati

```
item$ = "Jonathan Sea Gull"
CALL Insert((item$), "Livingston ", 10)
PRINT item$
```

il programma visualizzerebbe

```
Jonathan Livingston Sea Gull
Jonathan Sea Gull
```

Programma 6.16 *Passaggio per valore e passaggio per riferimento*

```
REM Passaggio per valore e passaggio per riferimento [6-16]
CALL Insert("Jonathan Sea Gull", "Livingston ", 10)
END

SUB Insert (first$, second$, spot)
  REM Inserisce second$ in first$, partendo da spot
  first$ = LEFT$(first$, spot - 1) + second$ + MID$(first$, spot)
  PRINT first$
END SUB

[Esecuzione]
Jonathan Livingston Sea Gull
```

Come si è visto, le funzioni e i sottoprogrammi hanno molte similitudini. A causa di questa somiglianza, d'ora in poi si utilizzerà la parola *procedura* per identificare questi due blocchi di enunciati.

COSTANTI SIMBOLICHE

QBasic può utilizzare due tipi di costanti: letterali e simboliche. Le costanti letterali, utilizzate nel corso di questo libro, appaiono come valori 'concreti' assegnati a delle variabili. Esempi di costanti letterali sono 5, 7.2E+23, 678& e "Ciao".

Le costanti simboliche sono anche conosciute come *costanti con nome*, dato che sono costanti a cui viene assegnato un nome. Se *elemento* fosse una costante numerica o stringa, l'enunciato

```
CONST NomeCostante = elemento
```

assegnerebbe il valore di *elemento* al nome *NomeCostante*. Si dice che la parola chiave CONST dichiara la costante simbolica *NomeCostante*.

Una costante simbolica dichiarata nella porzione principale di un programma viene riconosciuta da qualsiasi procedura. All'interno delle procedure, le costanti sono trattate come se venissero passate per valore. Infatti, se si prova a riassegnare un valore a una costante simbolica, sia con un enunciato di assegnazione che con il comando CONST, viene generato il messaggio di errore "Definizione doppia". Inoltre, una costante simbolica dichiarata all'interno di una procedura rimane locale per quella procedura. Ciò significa che non viene riconosciuta dalle istruzione del programma al di fuori della procedura in cui viene definita.

PASSAGGIO DI ARRAY ALLE PROCEDURE

Sono stati esaminati diversi esempi relativi al passaggio di una variabile numerica o stringa come argomento di una procedura. QBasic permette anche di passare un intero array. Per definire una funzione o un sottoprogramma in modo che possa ricevere un array, il parametro usato nell'enunciato FUNCTION o SUB deve contenere il nome dell'array seguito da una coppia di parentesi vuote. Inoltre, l'enunciato chiamante deve contenere il nome vero dell'array seguito da una coppia di parentesi vuote. Gli array vengono sempre passati per riferimento.

Nel Programma 6.17, il sottoprogramma MergeLists riunisce due elenchi ordinati di parole in un'unica lista. I parametri *listA$()*, *listB$()* e *newList$()* nella definizione del sottoprogramma indicano a QBasic di utilizzare questi array ogni volta che viene richiamato MergeLists. Nell'enunciato CALL, una serie di parentesi vuote segue ciascun nome di array. Le parentesi vuote consentono a QBasic di distinguere argomenti come *nome$* e *nome$()*, che devono essere interpretati, rispettivamente, come variabile stringa e array.

Programma 6.17 *Passaggio di array a una procedura*

```
REM Unisce due liste ordinate [6-17]
DIM oldNames$(1 TO 5), newNames$(1 TO 7), combinedList$(1 TO 12)
FOR index = 1 TO 5
  READ oldNames$(index)
NEXT index
FOR index = 1 TO 7
  READ newNames$(index)
NEXT index
CALL MergeLists(oldNames$(), newNames$(), combinedList$())
CLS
FOR index = 1 TO 12
  PRINT combinedList$(index); " ";
NEXT index
PRINT
DATA Alice, Beppe, Gino, Sandro, Tina
DATA Adamo, Bob, Ciro, Geppo, Kim, Marco, Stefano
END
```

```
SUB MergeLists (listA$(), listB$(), newList$())
  REM Ogni arrau deve essere dimensionato da un
  REM enunciato nella forma DIM nomeArray(1 TO m),
  REM dove m è un numero intero positivo
  sizeA = UBOUND(listA$)
  sizeB = UBOUND(listB$)
  sizeNew = UBOUND(newList$)
  REM Se l'array newList$() non è abbastanza grande per
  REM contenere le liste, non si deve iniziare.
  IF sizeNew < sizeA + sizeB THEN
      PRINT "L'array risultante non è abbastanza grande"
      EXIT SUB
  END IF
  aNow = 1
  bNow = 1
  newNow = 1
  DO WHILE (aNow <= sizeA) AND (bNow <= sizeB)
    IF listA$(aNow) < listB$(bNow) THEN
        newList$(newNow) = listA$(aNow)
        aNow = aNow + 1
      ELSE
        newList$(newNow) = listB$(bNow)
        bNow = bNow + 1
    END IF
    newNow = newNow + 1
  LOOP
  REM Uno dei due array, listA$() o listB$(),
  REM è terminato. Ora, solo uno dei due cicli
  REM seguenti svolgerà un'operazione.
  FOR index = aNow TO sizeA
    newList$(newNow) = listA$(index)
   newNow = newNow + 1
  NEXT index
  FOR index = bNow TO sizeB
    newNow = newNow + 1
  NEXT index
END SUB

[Esecuzione]
Adamo Alice Beppe Bob Ciro Geppo Gino Kim, Marco Sandro Stefano Tina
```

DICHIARAZIONE DEGLI ARRAY

Analogamente alle altre variabili, un array dichiarato in una procedura è, per default, locale e può essere reso statico o condiviso. A questo scopo, si devono includere le parole riservate STATIC o SHARED nell'enunciato DIM. La Figura 6.4 mostra l'inizio di un sottoprogramma che dichiara e dimensiona alcuni array.

Gli array locali sono sempre dinamici. Un array locale, come una qualsiasi altra variabile locale, viene impostato a zero, o come stringa nulla, ogni volta che viene richiamato il sottoprogramma, e viene rimosso dalla memoria al completamento dello stesso.

Se un enunciato FUNCTION o SUB è seguito dalla parola STATIC, un array statico dimensionato all'interno di una funzione o di un sottoprogramma conserverà i suoi valori tra una chiamata e l'altra. Si noti, tuttavia, che nonostante un array statico occupi continuamente una porzione di memoria anche dopo l'esecuzione del sottoprogramma o della funzione, non è possibile accedere all'array se non dal sottoprogramma o dalla funzione in cui è stato definito. Inoltre, ci si ricordi che un array statico deve essere dimensionato usando delle costanti al fine di specificare i limiti inferiore e superiore; non si possono usare delle variabili. Gli enunciati DIM per gli array locali e statici nelle procedure di tipo STATIC possono essere impartiti più volte senza causare un errore.

Gli array dimensionati nella porzione principale del programma possono essere condivisi con una particolare procedura includendo l'enunciato SHARED *nomeArray()* nella procedura, prima di usare l'array.

Nel sottoprogramma del Programma 6.18, l'array *count()* viene dimensionato come array locale.

Il Programma 6.19 mostra un semplice meccanismo di stack, conosciuto come LIFO (last-in-first-out, ultimo a entrare primo ad uscire). Questo stack consente al programma di memorizzare dei valori nell'ordine in cui vengono ricevuti e di prelevarli per l'elaborazione in ordine inverso. Nel sottoprogramma LifoStackControl, l'array *stack()* e la variabile *top%* devono essere caricate in una porzione di memoria permanente, e non devono essere create e inizializzate ogni volta che viene richiamato LifoStackControl. La parola STATIC alla fine dell'enunciato SUB fa in modo che tutte le variabili usate nel sottoprogramma conservino il loro valore tra le varie chiamate. Queste variabili vengono inizializzate una sola volta. L'impostazione delle variabili numeriche a zero e di quelle stringa come stringa nulla avviene solo la prima volta in cui viene richiamato il sottoprogramma.

```
SUB UsaAlcuniArray
   REM contatore() è una variabile locale
   DIM contatore(1 TO 20)
   DIM STATIC totali(1 TO 20)
   DIM SHARED ultimocont(1 TO 20)
   .
   .
END SUB
```

Figura 6.4 *Dichiarazione di un array in un procedura*

DIFFERENZE TRA LE FUNZIONI E I SOTTOPROGRAMMI

Tutte le operazioni svolte dalle funzioni possono essere eseguite anche dai sottoprogrammi. Tuttavia, quando l'obiettivo è quello di generare un solo risultato, le funzioni risultano più adatte allo scopo. Seguono le differenze principali tra le funzioni e i sottoprogrammi:

1. mentre i sottoprogrammi vengono richiamati dall'istruzione CALL, le funzioni possono essere usate nel punto in cui QBasic si aspetterebbe una costante o un'espressione;
2. mentre un nome di sottoprogramma serve solo per identificare il sottoprogramma, un nome di funzione identifica la funzione e l'enunciato all'interno della definizione le assegna un valore. Quando si richiama una funzione, viene generato un valore. Questo valore può essere una stringa o un qualsiasi tipo numerico, a seconda del tipo indicato dal nome della funzione;
3. le funzioni vengono generalmente utilizzate per calcolare un singolo valore.

Le prime due differenze sono illustrate dal Programma 6.20 che utilizza sia una funzione stringa che un sottoprogramma per invertire l'ordine dei caratteri in una frase digitata dall'utente.

NIDIFICAZIONE DI FUNZIONI E SOTTOPROGRAMMI

Come spiegato all'inizio del capitolo, un programmatore dovrebbe suddividere un problema in una sequenza di problemi più piccoli e meglio gestibili. In questo modo, si può risolvere ciascun sottoproblema con un sottoprogramma o una funzione. Quando l'operazione 'assegnata' a un sottoproblema risulta molto complessa, si dovrebbe suddividere ulteriormente il sottoproblema. Dato che QBasic consente di richiamare dei sottoprogrammi e delle funzioni da altrettanti sottoprogrammi e funzioni, è semplice codificare questo tipo di soluzione. Si dimostrerà ora questa procedura per risolvere un problema specifico.

Il calendario correntemente utilizzato, conosciuto come calendario Gregoriano, è stato introdotto nel 1582. Si supponga di voler convertire una qualsiasi data successiva all'anno 1582 in un formato che fornisca il giorno della settimana e mostri il mese in lettere. La Figura 6.5 mostra la suddivisione di questo problema in più sottoproblemi. Ogni rettangolo, ad eccezione di quello superiore, corrisponde a un sottoprogramma o a una funzione.

Programma 6.18 *Una procedura che utilizza un array locale*

```
REM Conta le occorrenze di alcune lettere in una stringa   [6-18]
INPUT "Inserire la stringa da analizzare: ", a$
CALL LetterCount(a$)
END

SUB LetterCount (info$)
  'a        contiene il codice ASCII di "A"
  'z        contiene il codice ASCII di "Z"
  'place    tiene traccia della posizione in info$
  'char     memorizza il codice ASCII del carattere
  '         in info$, e viene usato come indice nell'array
  '         count
  'count()  è un array che conta il numero di volte in cui
  '         una lettera appare in info$ (non viene
  '         considerata la differenza tra le maiuscole e
  '         le minuscole)
  a = ASC("A")
  z = ASC("Z")
  DIM count(a TO z)
  REM L'array dovrebbe inizialmente essere riempito con zeri
  REM QBasic lo fa automaticamente
  FOR place = 1 TO LEN(info$)
    char = ASC(UCASE$(MID$(info$, place, 1)))
    Se il carattere è una lettera la conta, altrimenti la ignora.
    IF char >= a AND char <= z THEN count(char) = count(char) + 1
  NEXT place
  REM Visualizza i risultati
  FOR char = a TO z
    PRINT USING " !"; CHR$(char);
  NEXT char
  PRINT
  FOR char = a TO z
    PRINT USING " #"; count(char);
  NEXT char
  PRINT
END SUB

[Esecuzione]
Inserire la stringa da analizzare: Il tempo e' denaro.
 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 1 0 0 1 3 0 0 0 1 0 0 1 1 1 2 1 0 1 0 1 0 0 0 0 0 0
```

Suddividendo il problema in più parti, la porzione principale del Programma 6.21 risulta concisa e ben comprensibile. Il sottoprogramma GetDate non richiede soltanto il mese, il giorno e l'anno, ma controlla che i dati forniti siano validi. I nomi usati per i sottoprogrammi e le funzioni richiamate da GetDate sono descrittivi.

Programma 6.19 *Un sottoprogramma con un array statico*

```
REM Dimostrazione del LIFO [6-19]
CALL LifoStackControl("push", 5, status$)
PRINT status$
CALL LifoStackControl("push", 8, status$)
PRINT status$
CALL LifoStackControl("pop", num, status$)
PRINT status$; num
CALL LifoStackControl("pop", x, status$)
PRINT status$; x
'Tutti i valori inseriti sono stati estratti, quindi status$
'riporta eventuali errori causati da un'altra estrazione.
CALL LifoStackControl("pop", wrong, status$)
PRINT status$; wrong
END

SUB LifoStackControl (operation$, value, status$) STATIC
  REM Implementazione di un semplice LIFO
  DIM stack(1 TO 256)
  'operation$ azione da eseguire, PUSH o POP
  'value      il dato da inserire nello stack o il dato
  '           da estrarre dallo stack
  'status$    passa "OK" se l'operazione ha avuto successo,
  '           o un messaggio di errore in caso contrario
  'top%       registra l'ultima locazione in cui sono
  '           stati inseriti i dati
  'stack      è un array statico che contiene i valori che
  '           vengono inseriti o estratti
  SELECT CASE UCASE$(operation$)
    CASE "PUSH"
      IF top% < 256 THEN
          top% = top% + 1
          stack(top%) = value
          status$ = "OK"
        ELSE
          status$ = "Lo stack è pieno"
      END IF
    CASE "POP"
      IF top% > 0 THEN
          value = stack(top%)
          top% = top% - 1
          status$ = "OK"
        ELSE
          status$ = "Lo stack è vuoto"
      END IF
    CASE ELSE
      status$ = "Errore di stack"
  END SELECT
END SUB
```

```
[Esecuzione]
OK
OK
OK
OK 8
OK 5
Lo stack è vuoto
```

Programma 6.20 *Differenze tra le funzioni e i sottoprogrammi*

```
'Mostra la differenza tra una funzione e un sottoprogramma [6-20]
'invertendo i caratteri in una stringa.
INPUT "Inserire una frase: ", phrase$
PRINT "  "; RevLine$(phrase$)
CALL ReverseLine(phrase$, answer$)
PRINT "  "; answer$
END

SUB ReverseLine (info$, result$)
  FOR index = LEN(info$) TO 1 STEP -1
    result$ = result$ + MID$(info$, index, 1)
  NEXT index
END SUB

FUNCTION RevLine$ (info$)
  FOR index = LEN(info$) TO 1 STEP -1
    temp$ = temp$ + MID$(info$, index, 1)
  NEXT index
  RevLine$ = temp$
END FUNCTION

[Esecuzione]
Inserire una frase: Chi la fa l'aspetti
  ittepsa'l af al ihC
  ittepsa'l af al ihC
```

Si noti che a DateIsValid viene assegnato il risultato di un'espressione logica, che può essere vero o falso. È meglio pensare il valore della funzione DateIsValid come vero o falso, piuttosto che come valore 0 o -1.

La funzione DateIsValid richiama inoltre la funzione IsALeapYear che è stata introdotta nel Programma 6.10. DateIsValid dimostra che una funzione non si limita a calcolare semplicemente un valore, ma può svolgere anche altre operazioni. Parte del lavoro di DateIsValid consiste nel visualizzare dei messaggi di errore nel caso in cui l'utente fornisca una data non valida.

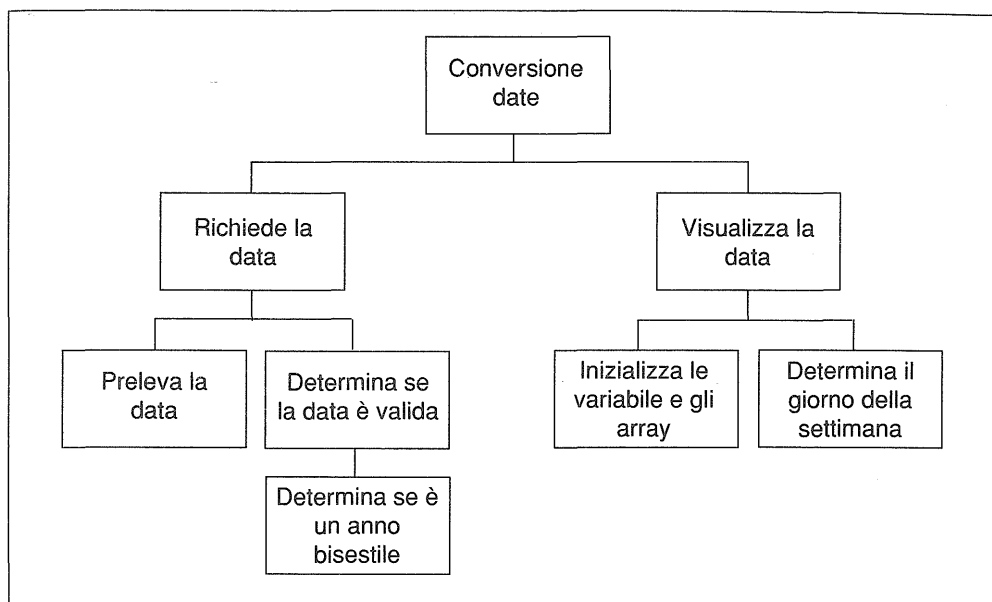



Figura 6.5 *Diagramma del flusso logico dell'analisi*

Il sottoprogramma PrintFancyDate dimensiona due array locali, uno per contenere i nomi dei giorni della settimana e l'altro per i nomi dei mesi, e richiama quindi il sottoprogramma Initialize per assegnare dei valori agli array. Inserendo queste assegnazioni in un altro sottoprogramma, si evita di sovraffollare il sottoprogramma PrintFancyDate. Si note inoltre il modo conciso e leggibile in cui PrintFancyDate richiama la funzione DayOfWeek. Dato che QBasic consente di utilizzare una funzione in qualsiasi punto in cui sia ammessa una costante, si può tranquillamente ottenere l'indice dell'array day$ richiamando la funzione DayOfWeek.

La funzione DayOfWeek determina il giorno della settimana che corrisponde alla data specificata tramite una formula apparentemente strana, ma molto efficace. Si noti che, per fare in modo che la formula possa funzionare, il programma considera i mesi gennaio e febbraio come mesi 13 e 14 dell'anno precedente.

Programma 6.21 *Un programma con funzioni e sottoprogrammi nidificati*

```
REM Converte una data numerica in giorno, mese, anno  [6-21]
CALL GetDate(month, day, year)
CALL PrintFancyDate(month, day, year)
END

FUNCTION DateIsValid (month, day, year)
  yearOk = (year >= 1582)
```

```
  monthOk = (month >= 1) AND (month <= 12)
  SELECT CASE month
    CASE 2
      IF ThisIsALeapYear(year) THEN
          dayOk = (day >= 1) AND (day <= 29)
        ELSE
          dayOk = (day >= 1) AND (day <= 28)
      END IF
    CASE 4, 6, 9, 11
      dayOk = (day >= 1) AND (day <= 30)
    CASE ELSE
      dayOk = (day >= 1) AND (day <= 31)
  END SELECT
  DateIsValid = yearOk AND monthOk AND dayOk
  IF NOT (yearOk AND monthOk AND dayOk) THEN
    PRINT "La data non è valida. Premi un tasto per continuare."
    temp$ = INPUT$(1)
  END IF
END FUNCTION

FUNCTION DayOfWeek (m, d, y)
  REM m = mese, d = giorno, y = anno
  IF m <= 2 THEN
     m = m + 12
     year = year - 1
  END IF
  DayOfWeek = (d + 2 * m + 3 * (m + 1) \ 5 + y + y \ 4 - y \ 100 + y_
             \ 400 + 2) MOD 7
END FUNCTION

SUB GetDate (month, day, year)
  DO
    CALL RequestDate(month, day, year)
  LOOP UNTIL DateIsValid(month, day, year)
END SUB

SUB Initialize (d$(), m$())
  d$(0) = "Sabato":      d$(1) = "Domenica"
  d$(2) = "Lunedì":      d$(3) = "Martedì"
  d$(4) = "Mercoledì":   d$(5) = "Giovedì"
  d$(6) = "Venerdì"
  m$(1) = "Gennaio":     m$(2) = "Febbraio"
  m$(3) = "Marzo":       m$(4) = "Aprile"
  m$(5) = "Maggio"       m$(6) = "Giugno"
  m$(7) = "Luglio":      m$(8) = "Agosto"
  m$(9) = "Settembre":   m$(10) = "Ottobre"
  m$(11) = "Novembre":   m$(12) = "Dicembre"
END SUB
```

```
SUB PrintFancyDate (m, d, y)
  DIM day$(0 TO 6), month$(1 TO 12)
  CALL Initialize(day$(), month$())
  IF d < 10 THEN
      f$ = "&_, & #_, ####"
    ELSE
      f$ = "&_, & ##_, ####"
   END IF
  PRINT USING f$; day$(DayOfWeek(m, d, y)); month$(m); d; y
END SUB

SUB RequestDate (month, day, year)
  CLS
  INPUT "Mese (1-12)"; month
  INPUT "Giorno (1-31)"; day
  INPUT "Anno (1582-->)"; year
END SUB

FUNCTION ThisIsALeapYear (year)
  true = -1: false = 0
  IF year MOD 4 <> 0 THEN
      ThisIsALeapYear = false
    ELSEIF (year MOD 100 = 0) AND (year MOD 400 <> 0) THEN
      ThisIsALeapYear = false
    ELSE
      ThisIsALeapYear = true
  END IF
END FUNCTION

[Esecuzione]
Mese (1-12)? 7
Giorno (1-31)? 4
Anno (1582-->)? 1776
Giovedì, Luglio 4, 1776
```

Gli utenti che hanno scritto dei programmi con dei linguaggi che permettono di nidificare le definizioni dei sottoprogrammi e delle funzioni, dovrebbero notare che QBasic non ammette questo stile di programmazione; è necessario definire ogni sottoprogramma e funzione separatamente.

RICORSIVITÀ

Una procedura, oltre a poter richiamare altre procedure, può richiamare se stessa. Questa operazione è chiamata *ricorsività.* Ci sono alcuni problemi che per essere risolti necessitano di operazioni ricorsive. Ad esempio, si supponga che in uno stato di 6 milioni di abitanti sia in atto una crescita di popolazione annua del 2% e un'immigrazione di 20.000 persone all'anno. Per ottenere la popolazione dello stato

all'inizio di un anno sulla base della popolazione dell'anno precedente, si può usare questa semplice formula :

```
(pop_inizio_anno)=1.02*(pop_anno_precedente) +20000
```

Dato che 2% equivale a 0,02, si può pensare al numero 1,02 come 1+0,02, oppure 1 più 2%. Quindi, la moltiplicazione della popolazione per 1,02 equivale all'aggiunta del 2% alla popolazione corrente. A questo punto, si definisca la funzione Popolazione(*n*) per ottenere la popolazione dello stato all'inizio di un qualsiasi anno. Il parametro *n* presume valori come 0, 1, 2 e così via, che corrispondono, rispettivamente, all'anno corrente, a quello successivo eccetera. Se Popolazione(0) contiene il valore 6.000.000, si può usare la formula appena scritta per calcolare la popolazione dello stato negli anni successivi:

```
Popolazione(1) = 1.02*Popolazione(0) + 20000
Popolazione(2) = 1.02*Popolazione(1) + 20000
.
.
Popolazione(n) = 1.02*Popolazione(n-1) + 20000
```

Si supponga di voler sapere il valore di Popolazione(3). Si potrebbero eseguire i calcoli seguenti:

```
Popolazione(3) = 1.02*Popolazione(2) + 20000
```

Ora,

```
Popolazione(2) = 1.02*Popolazione(1) + 20000
```

e

```
Popolazione(1) = 1.02*Popolazione(0) + 20000
```

Si sa, invece, che Popolazione(0) è uguale a 6.000.000. Per ottenere il valore di Popolazione(3) si possono eseguire le varie equazioni una alla volta:

```
Popolazione(1) = 1.02*6000000 + 20000 = 6140000

Popolazione(2) = 1.02*6140000 + 20000 = 6282800

Popolazione(3) = 1.02*6282800 + 20000 = 6428456
```

Il Programma 6.22 segue questa procedura. Ogni volta che viene richiamata la funzione Population&, il programma decrementa il valore di *n%* di 1. Quando il valore di *n%* raggiunge lo zero, la ricorsività termina. Ogni volta che si utilizza una procedura ricorsiva, ci si deve *assicurare* che esista un modo per interrompere la ricorsività. In caso contrario, QBasic mostra un messaggio di errore e non procede con l'esecuzione.

Programma 6.22 *Una funzione ricorsiva*

```
REM Una funzione ricorsiva [6-22]
INPUT "Numero di anni"; numberOfYears%
PRINT "La popolazione sarà"; Population&(numberOfYears%)
END

FUNCTION Population& (n%)
  IF n% = 0 THEN
      Population& = 6000000
    ELSE
      Population& = 1.02 * Population&(n% - 1) + 20000
  END IF
END FUNCTION

[Esecuzione]
Numero di anni? 3
La popolazione sarà 6428456
```

Il Programma 6.23 utilizza un sottoprogramma ricorsivo per indovinare il numero pensato da una persona. La porzione principale del programma fornisce alcune direttive all'utente, richiama il sottoprogramma Guess e consente di ripetere il gioco.

Il sottoprogramma Guess inizia visualizzando un numero compreso tra il valore più alto e quello più basso di quelli che possono essere pensati dall'utente. Inizialmente, questo valore è compreso tra 0 e 1000. Guess richiede quindi di specificare se il numero visualizzato è maggiore, minore o uguale a quello pensato. Se il numero è maggiore, il programma esegue un altro tentativo visualizzando un numero più basso di quello precedente. Analogamente, se il numero è minore, viene visualizzato un valore più alto. Questa procedura continua fino a quando il numero non viene indovinato. Guess tiene inoltre il conto dei tentativi effettuati a scopo statistico.

Grazie a tutti questi strumenti si possono risolvere dei problemi molto complessi. Dei problemi complessi, tuttavia, richiamano dati più complessi. Le strutture per gestire questi tipi di dati sono presentate nel prossimo capitolo.

Programma 6.23 *Una procedura ricorsiva*

```
REM Indovina il numero [6-23]
REM L'utente pensa un numero, il computer lo indivina
DO
  PRINT "Pensa un numero tra 1 e 1000, "
  PRINT "ma non dirmi qual è."
  PRINT "Indovinerò il numero in 10 tentativi o meno."
  PRINT "Premi un tasto per iniziare."
  response$ = INPUT$(1)
  tries = 1
  CALL Guess(1, 1000, tries)
  PRINT "Vuoi giocare ancora (S,N)?"
  response$ = UCASE$(INPUT$(1))
  CLS
LOOP UNTIL response$ = "N"
END

SUB Guess (Low, High, tries)
  currentGuess = INT((Low + High) / 2)
  PRINT
  PRINT USING "Io dico ####!"; currentGuess; ".";
  PRINT "Il mio tentativo è Alto, Basso o Giusto (A,B,G)?"
  DO
    response$ = UCASE$(INPUT$(1))
  LOOP UNTIL (response$ = "A") OR (response$ = "B") OR (response$ = "G")
  SELECT CASE response$
    CASE "A"
      High = currentGuess - 1
      CALL Guess(Low, High, tries + 1)
    CASE "B"
      Low = currentGuess + 1
      CALL Guess(Low, High, tries + 1)
    CASE "G"
      PRINT
      PRINT "Ho indovinato il numero in"; tries; "tentativi."
  END SELECT
END SUB

[Esecuzione]
Pensa un numero tra 1 e 1000,
ma non dirmi qual è.
Indovinerò il numero in 10 tentativi o meno.
Premi un tasto per iniziare.

Io dico 500
Il mio tentativo è Alto, Basso o Giusto (A,B,G)? A
Io dico 250
Il mio tentativo è Alto, Basso o Giusto (A,B,G)? B
```

```
Io dico 375
Il mio tentativo è Alto, Basso o Giusto (A,B,G)? G

Ho indovinato il numero in 3 tentativi.
Vuoi giocare ancora (S,N)? N
```

CAPITOLO 7

FILE DI DATI

Nei capitoli precedenti, i dati elaborati da un programma sono stati specificati tramite l'istruzione LET, memorizzati in enunciati DATA, o forniti dall'utente in risposta ai comandi INPUT. Questi metodi sono sufficienti per piccole quantità di dati da usare in un solo programma. Tuttavia, delle grosse quantità di dati, dei dati a cui devono accedere diversi programmi, o dati che devono essere aggiornati dall'utente, devono essere conservati su disco.

QBasic offre tre modi differenti per organizzare i dati in file su disco. I tre tipi di file risultanti sono chiamati sequenziali, ad accesso casuale e binari. Ciascun tipo di file comporta dei vantaggi e degli svantaggi. I file sequenziali utilizzano lo spazio in modo più efficiente, ma non sono facili da aggiornare e da utilizzare per la ricerca delle informazioni. I file ad accesso casuale forniscono un accesso rapido ai dati, ma risultano più difficile da gestire e mantenere da programma. I file binari, infine, offrono grande flessibilità, ma dato che non dispongono di una struttura, è compito del programmatore assicurarsi che i dati vengano interpretati correttamente.

Questo capitolo discute la creazione e l'uso di tutti questi tipi di file. La procedura di creazione consente di memorizzare fisicamente i dati sul disco. Il computer può prelevare successivamente questi dati e assegnarli a delle variabili, in un modo molto simile a quello esaminato per l'enunciato DATA.

FILE SEQUENZIALI

La tabella ASCII nell'Appendice A contiene 256 caratteri. Non tutti questi caratteri possono essere visualizzati sullo schermo o stampati su carta, ma tutti possono essere utilizzati nei file. I caratteri con codice ASCII compreso tra 0 e 31 sono chiamati caratteri di controllo.

I caratteri di controllo con codice 10 e 13, che rappresentano rispettivamente l'avanzamento riga e il ritorno a capo, hanno un significato particolare nei file sequenziali. La coppia di caratteri composta da un carattere di ritorno a capo e da uno di avanzamento riga viene scritta CR/LF.

La coppia CR/LF è il metodo standard per identificare la fine di una riga. Per esempio, per indicare la fine di una riga in QBasic si preme il tasto Invio. Questo tasto invia una coppia CR/LF che sposta il cursore sullo schermo all'inizio della riga successiva.

Si può pensare a un file sequenziale come a una lunga sequenza di caratteri. I file sequenziali contengono normalmente un certo numero di coppie CR/LF che suddividono il file in blocchi denominati *record*. Nella maggior parte dei file sequenziali discussi in questa sezione, ogni record consiste di alcuni gruppi di dati tra loro relazionati, chiamati *campi*, che sono separati da virgole.

Il file sequenziale riportato in Figura 7.1 contiene delle informazioni relative agli impiegati di una società. Il record per ogni impiegato contiene quattro campi: uno per il nome dell'impiegato, uno per un numero di identificazione, uno per il costo orario e l'ultimo per l'importo dovuto. Ogni stringa appare tra virgolette e ogni valore numerico è espresso come intero. Se questo file venisse visualizzato sullo schermo tramite il comando TYPE del DOS, si otterrebbe un output come quello riportato in Figura 7.2.

CREAZIONE DI UN FILE SEQUENZIALE

Ci sono diversi modi per organizzare e inserire i dati in un file sequenziale. La tecnica presentata in questa sede è facile da riportare e utilizzare. Le altre tecniche vengono discusse più avanti in questo stesso capitolo.

```
"Mario,Rossi","123-45-6789",17000,19091000<CR/LF>"Alberto,Rubinetti",
"456-98-7654",15000,1537650<CR/LF>"Davide,Villa","238-91-2355",25000,
934700<CR/LF>
```

Figura 7.1 *Un file sequenziale*

1. Scegliere un nome di file. Ci si ricordi che un nome di file è una stringa che consiste di due parti: un nome composto al massimo da 8 caratteri e un'estensione, opzionale, che consiste di un punto e un massimo di tre caratteri. Si possono utilizzare lettere, numeri e i simboli &!_@"~(){}-#%$ sia nel nome che nell'estensione. Alcuni esempi di nome di file sono SPESE_87, CLIENTI.DAT e FATTURE.91;
2. scegliere un numero compreso tra 1 e 255 come *numero di riferimento* del file. In QBasic, un file viene identificato da questo numero;
3. eseguire l'enunciato

   `OPEN nomefile FOR OUTPUT AS #n`

   dove *n* rappresenta il numero di riferimento. Questo enunciato può essere interpretato come *apertura di un file per l'output.* Questa procedura apre un canale di comunicazione tra il computer e il disco e consente di memorizzare dei dati nel file specificato.

Attenzione: Si dovrebbe impartire questo comando una sola volta quando si vuole creare il file. Se si apre un file già esistente con questo enunciato, il computer cancella i dati in esso contenuti. Si vedrà tra breve come modificare un file esistente.

4. Memorizzare i dati nel file tramite il comando WRITE#. Se a$ è una stringa, l'enunciato

   `WRITE #n, a$`

   scrive la stringa *a$* nel file racchiudendola tra virgolette. Ci si ricordi che *n* rappresenta il numero di riferimento del file. Se *c* è un numero, l'enunciato

   `WRITE #n, c`

   scrive il numero *c*, senza spazi in testa e in coda, nel file numero *n*. L'enunciato

   `WRITE #n, a$, c`

   inserisce *a$* e *c* separandoli con una virgola. Analogamente, se alcune stringhe, numeri, o una combinazione di entrambi seguono il comando WRITE #*n*, tutte le stringhe e i numeri vengono inseriti nel file e sono separati da virgole. Dopo l'esecuzione di ciascun enunciato WRITE #, QBasic aggiunge una coppia di CR/LF nel file;

```
"Mario,Rossi","123-45-6789",17000,19091000
"Alberto,Rubinetti","456-98-7654",15000,1537650
"Davide,Villa","238-91-2355",25000,934700
```

Figura 7.2 *Un file sequenziale visualizzato con il comando TYPE del DOS*

5. dopo aver memorizzato tutti i dati nel file, impartire il comando

   ```
   CLOSE #n
   ```

   dove *n* rappresenta il numero di riferimento del file. Questa operazione rimuove l'assegnazione di questo numero e indica al DOS la fine del file. Se si omette il numero di riferimento, vengono chiusi tutti i file aperti.

Il Programma 7.1 crea il file riportato in Figura 7.1. I valori nell'ultimo enunciato DATA vengono chiamati *valori sentinella* e segnalano al computer la fine dei dati.

AGGIUNTA DI DATI A UN FILE SEQUENZIALE

Per aggiungere dei dati alla fine di un file sequenziale, si proceda nel modo seguente:

1. scegliere un numero compreso tra 1 e 255 come numero di riferimento del file. Questo numero non deve necessariamente essere uguale a quello utilizzato al momento della creazione;
2. eseguire l'enunciato

   ```
   OPEN nomefile FOR APPEND AS #n
   ```

   dove *n* rappresenta il numero di riferimento. Con questo comando si apre il file per l'aggiunta dei dati. L'enunciato indica al computer di registrare i dati alla fine del file specificato. Se il file specificato non esiste, ne viene creato uno nuovo;
3. inserire i dati nel file utilizzando l'enunciato WRITE#;
4. dopo aver memorizzato tutti i dati nel file, chiudere il file con l'istruzione CLOSE #n.

Il Programma 7.2 aggiunge dei nuovi record alla fine del file PAYROLL.91. La Figura 7.3 mostra il contenuto di questo file dopo l'esecuzione del Programma 7.2.

LETTURA DI DATI DA UN FILE SEQUENZIALE

I dati memorizzati in un file sequenziali possono essere letti e assegnati a delle variabili. A questo scopo, si proceda nel modo seguente:

1. scegliere un numero compreso tra 1 e 255 come numero di riferimento del file. Questo numero non deve necessariamente essere uguale a quello utilizzato al momento della creazione;

Programma 7.1 *Creazione di un file sequenziale*

```
REM Creazione del file PAYROLL.91 e inserimento di dati. [7-1]
OPEN "PAYROLL.91" FOR OUTPUT AS #1
READ nom$, ssn$, hourlyWage%, yearToDate&
DO UNTIL nom$ = "EOD"        'Continua fino alla fine dei dati
  WRITE #1, nom$, ssn$, hourlyWage%, yearToDate&
  READ nom$, ssn$, hourlyWage%, yearToDate& 'Legge i dati successivi
LOOP
CLOSE #1
REM -- Dati:nome, numero, costo orario, importo dovuto
DATA "Mario,Rossi", 123-45-6789, 17000, 19091000
DATA "Alberto,Rubinetti", 456-98-7654, 15000, 1537650
DATA "Davide,Villa", 238-91-2355, 25000, 934700
DATA EOD, "", 0, 0
END
```

Programma 7.2 *Aggiunta di nuovi record al file PAYROLL.91*

```
REM Inserisce un altro record nel file PAYROLL.91  [7-2]
OPEN "PAYROLL.91" FOR APPEND AS #1
CLS
INPUT "Nome"; firstName$
INPUT "Cognome"; lastName$
INPUT "Numero di identificazione"; ssn$
INPUT "Costo orario"; hourlyWage%
INPUT "Importo dovuto"; yearToDate&
nom$ = firstName$ + "," + lastName$
WRITE #1, nom$, ssn$, hourlyWage%, yearToDate&
CLOSE #1
END

[Esecuzione]
Nome? Andrea
Cognome? Gerardi
Numero di identificazione? 450-21-3678
Costo orario? 16000
Importo dovuto? 1275900
```

```
"Mario,Rossi","123-45-6789",17000,19091000<CR/LF>"Alberto,Rubinetti",
"456-98-7654",15000,1537650<CR/LF>"Davide,Villa","238-91-2355",25000,
934700<CR/LF>"Andrea,Gerardi","450-21-3678",16000,1275900<CR/LF>
```

Figura 7.3 *Il contenuto del file PAYROLL.91 dopo l'aggiornamento*

2. eseguire l'enunciato

```
OPEN nomefile FOR INPUT AS #n
```

dove *n* rappresenta il numero di riferimento. Con questo comando si apre il file per la lettura dei dati. L'enunciato apre un canale di comunicazione tra il computer e il disco e consente di leggere dei dati dal file specificato;

3. leggere i dati dal file tramite il comando INPUT#. L'enunciato INPUT# assegna i dati prelevati dal file a delle variabili ed opera in modo simile al comando INPUT che assegna i dati letti da tastiera. Per utilizzare correttamente il comando INPUT# è necessario sapere come è stata utilizzata l'istruzione WRITE# per registrare i dati. Ciò significa che bisogna conoscere l'ordine con cui sono stati memorizzati i dati nel file, al fine di poter inserire i dati nelle variabili appropriate. L'enunciato

```
INPUT #n, var1, var2, ...
```

assegna a ciascuna variabile un elemento del file. I dati nel file sono separati da virgole o dalla coppia CR/LF. Il numero e il tipo delle variabili specificate nel comando INPUT# devono essere uguali a quelli utilizzati al momento della registrazione dei dati;

4. dopo aver letto i dati desiderati, chiudere il file tramite l'enunciato CLOSE #*n*.

Programma 7.3 *Calcolo delle ore di lavoro di ciascun impiegato*

```
REM Calcolo delle ore di lavoro di ciascun impiegato [7-3]
OPEN "PAYROLL.91" FOR INPUT AS #1
CLS
PRINT "Nome"; TAB(20); "Ore di lavoro"
PRINT "---------------"; TAB(20); "-------------"
DO UNTIL EOF(1)          'Elabora tutto il file
  INPUT #1, nom$, ssn$, hourlyWage%, yearToDate&
  PRINT nom$; TAB(20);
  PRINT USING "    #####"; yearToDate& / hourlyWage%
LOOP
CLOSE #1
END

[Esecuzione]
Nome                Ore di lavoro
---------------     -------------
Mario,Rossi              1128
Alberto,Rubinetti         102
Davide,Villa               37
Andrea,Gerardi             79
```

QBasic mette a disposizione la funzione EOF che consente di determinare se è stata raggiunta la fine del file. In qualsiasi momento, la condizione

```
EOF(n)
```

è vera se è stata raggiunta la fine del file il cui numero di riferimento è uguale a *n*. In caso contrario, la condizione è falsa.

Il Programma 7.3 restituisce il numero di ore di lavoro di ciascun impiegato in base ai dati contenuti nel file PAYROLL.91. Utilizzando l'enunciato DO UNTIL EOF(1), il programmatore non deve sapere il numero di record contenuti nel file. Il formato PRINT USING impedisce la visualizzazione della parte decimale eventualmente generata dalla divisione.

La Funzione EOF ricopre lo stesso ruolo dei valori sentinella (come EOD o -1) negli enunciati DATA. Ci sono tuttavia alcune differenze importanti. La Figura 7.4a mostra un tipico segmento di programma che legge un record prima di entrare in un ciclo e ne legge un altro alla fine del ciclo. Questa costruzione è necessaria perché QBasic rileva una condizione di fine dati leggendo un valore sentinella estraneo che non deve essere elaborato.
Un programma che legge i dati da un file, invece, deve verificare la condizione di fine file *prima*, e *poi* prelevare un record dal file, come mostrato in Figura 7.4b. Questa costruzione è necessaria perché la condizione di fine file è vera *solo nel momento in cui* QBasic rileva che non ci sono più dati nel file. L'esecuzione del comando INPUT# quando è stata incontrata la fine del file genera un errore.

Il Programma 7.4 illustra un errore molto comune che si verifica nella lettura di un file sequenziale. Il programma non elabora l'ultimo elemento nel file. Inoltre, se il file non contiene dati, la terza riga genera il messaggio di errore "Input oltre la fine del file".

I file sequenziali possono essere molto estesi. Invece di listare l'intero contenuto, si possono cercare determinate informazioni. Il Programma 7.5 richiede all'utente il numero di identificazione e trova l'impiegato corrispondente nel file PAYROLL.91, esaminando ciascun record e trovando quello desiderato. Dato che il numero specificato potrebbe non esistere, è necessario utilizzare la funzione EOF() per determinare la condizione di fine file.

```
READ info$, eccetera
DO UNTIL info$ = "EOD"           DO UNTIL EOF(1)
   'elabora info$, eccetera         INPUT #1, info$, eccetera
      .                             'elabora info$, eccetera
                                       .
      .
   READ info$, eccetera                .
LOOP                             LOOP
```

Figura 7.4 *Cicli READ/DATA e INPUT#*

Programma 7.4 *Un programma con un errore*

```
REM Un programma con un errore [7-4]
OPEN "PAYROLL.91" FOR INPUT AS #1
CLS
INPUT #1, nom$, ssn$, hourlyWage%, yearToDate&
DO UNTIL EOF(1)
  IF hourlyWage% > 1000 THEN PRINT nom$
  INPUT #1, nom$, ssn$, hourlyWage%, yearToDate&
LOOP
CLOSE #1
END
```

Programma 7.5 *Una ricerca in un file sequenziale*

```
REM Trova un impiegato usando il numero di identificazione  [7-5]
OPEN "PAYROLL.91" FOR INPUT AS #1
CLS
INPUT "Numero di identificazione"; a$
ssn$ = ""
DO UNTIL (ssn$ = a$) OR EOF(1)    'Esamina il file
  INPUT #1, nom$, ssn$, hourlyWage%, yearToDate&
LOOP
IF ssn$ = a$ THEN
    PRINT nom$; " ha il numero "; a$
  ELSE
    PRINT a$; " non esiste nel file PAYROLL.91"
END IF
CLOSE #1
END

[Esecuzione]
Numero di identificazione? 222-33-4444
222-33-4444 non esiste nel file PAYROLL.91
```

ALTRI METODI PER INSERIRE E PRELEVARE DEI DATI DA UN FILE SEQUENZIALE

Oltre agli enunciati WRITE# e INPUT#, esistono altri comandi che consentono di memorizzare e prelevare dei dati da un file sequenziale.

Le istruzioni PRINT# e PRINT# USING inseriscono dei dati in un file operando in modo analogo ai comandi PRINT e PRINT USING che visualizzano delle informazioni sullo schermo. Se si utilizza PRINT#, QBasic memorizza i numeri con degli spazi in coda ed eventualmente in testa. Dei punti e virgola usati come separatori consentono

di visualizzare successivamente i dati uno dopo l'altro, mentre delle virgole attivano le zone. A meno che alla fine del comando PRINT# o PRINT# USING non si inserisca un punto e virgola o da una virgola, QBasic memorizza automaticamente una coppia CR/LF.

Gli enunciati LINE INPUT# e INPUT$ possono essere usati per leggere i dati da un file sequenziale. L'enunciato

```
LINE INPUT #n, a$
```

legge tutti i caratteri dalla posizione del puntatore fino alla coppia CR/LF successiva nel file identificato dal numero di riferimento *n*, e li assegna alla variabile *a$*. L'enunciato

```
a$ = INPUT$(m, n)
```

legge gli *m* caratteri successivi dal identificato dal numero di riferimento *n*, e li assegna alla variabile *a$*. Questo enunciato legge tutti i caratteri, compresi i punti e virgola, le virgole e le coppie CR/LF. LINE INPUT# viene normalmente usato per leggere i dati registrati con il comando PRINT#.

ORDINAMENTO DI UN FILE SEQUENZIALE

Oltre ad accedere ai file sequenziali per prelevare delle informazioni, si possono modificare, rimuovere e aggiungere dei nuovi record. Queste operazioni possono essere svolte in modo più efficiente se prima si ordinano i dati nel file. Per ordinamento, si intende ridisporre i record del file in un ordine logico. A questo scopo, è necessario scegliere un campo su cui basare l'ordinamento.

I record di un file possono essere ordinati sulla base di qualsiasi campo, caricando i dati in alcuni array (un array per campo) e utilizzando il comando SWAP mentre si ordina l'array che contiene il campo che si vuole usare come chiave di ordinamento. Il Programma 7.6 si serve di questa tecnica per ordinare il file sequenziale PAYROLL.91 sulla base del nome degli impiegati.

Se si ricordano le direttive esposte nel Capitolo 6, si può notare che il Programma 7.6 costituisce un ottimo esempio di programma ben strutturato. La porzione principale è concisa e consiste principalmente di chiamate a sottoprogrammi il cui nome descrive le operazioni da essi svolte. Il sottoprogramma CountRecords determina la quantità di record presenti nel file PAYROLL.91 al fine di dimensionare degli array sufficientemente capaci per contenere tutti i dati. Viene definito un array per ciascun campo, e gli array con lo stesso indice contengono i campi dello stesso record. Il programma, inoltre, consente di velocizzare le operazioni offrendo all'utente l'opportunità di specificare il numero di record del file, nel caso sia conosciuto.

Se l'utente non conosce questo numero, il programma utilizza l'enunciato LINE INPUT# per leggere tutti i caratteri fino alla coppia CR/LF successiva; in questo modo, viene letto un intero record senza considerare i singoli campi. Il numero di enunciati LINE INPUT# eseguiti prima della fine del file costituisce il numero di record presenti nel file.

Il sottoprogramma LoadArrays usa il comando INPUT# per leggere i quattro campi di ciascun record direttamente negli elementi appropriati dell'array. Viene usato il ciclo FOR i = 1 TO count...NEXT i invece del ciclo DO UNTIL EOF(1)...LOOP, dato che il programma conosce già il numero di record da elaborare. Inoltre, se l'utente se l'utente avesse fornito un numero sbagliato di record e il programma usasse il ciclo DO UNTIL, si correrebbe il rischio di leggere più dati di quanti ne possano contenere gli array.

Il sottoprogramma SortByName usa un tipo di ordinamento particolare conosciuto come *bubble sort* (ordinamento a bolle). Con questo tipo di ordinamento, i valori 'più alti galleggiano' nella parte superiore del file. La procedura di ordinamento confronta i valori nell'array. Quando ne vengono rilevati due non ordinati correttamente, il programma ne scambia la posizione. Ogni volta che viene cambiata la posizione di due elementi dell'array *nom$*, è necessario scambiare anche gli elementi corrispondenti degli altri tre array. In questo modo si è sicuri che gli elementi degli array con lo stesso indice contengano sempre le informazioni relative allo stesso record.

Il sottoprogramma WriteArrays esegue l'operazione opposta di LoadArrays. WriteArrays utilizza l'enunciato WRITE# per inserire i dati dall'array nel file su disco. Come accade negli altri sottoprogrammi, WriteArrays visualizza un messaggio prima di iniziare la procedura. Questo messaggio informa l'utente che la procedura di ordinamento è avvenuta correttamente.

Esiste un altro metodo per ordinare un file usando, come chiave di ordinamento, il primo campo. Il DOS dispone del comando SORT che permette di ordinare i record di un file sequenziale. Il comando

```
SORT <nomefile1> nomefile2
```

ordina i record di *nomefile1* usando il primo campo come chiave di ordinamento e li scrive in un nuovo file denominato *nomefile2*. Il comando DOS

```
SORT <nomefile1> nomefile2 /R
```

usa un ordinamento decrescente.

Il Programma 7.7 produce lo stesso risultato del programma 7.6. La seconda riga cambia il nome del file PAYROLL.91. L'enunciato SHELL richiama il DOS ed impartisce il comando SORT.

Programma 7.6 *Ordinamento del file PAYROLL.91*

```
REM Ordina i record per nome di impiegato  [7-6]
DEFINT A-Z    'definisce le variabili senza tipo come intere
CLS
CALL CountRecords(total)
DIM nom$(total), ssn$(total), hourlyWage%(total), yrToDate&(total)
CALL LoadArrays(total, nom$(), ssn$(), hourlyWage%(), yrToDate&())
CALL SortByName(total, nom$(), ssn$(), hourlyWage%(), yrToDate&())
CALL WriteArrays(total, nom$(), ssn$(), hourlyWage%(), yrToDate&())
PRINT "Ordinamento completo"
END

SUB CountRecords (number)
  PRINT "Quanti record ci sono nel file PAYROLL.91?"
  INPUT "(Inserire 0 per lasciare questo compito al computer.) ",
number
  IF number = 0 THEN
    PRINT "Conteggio dei record"
    OPEN "PAYROLL.91" FOR INPUT AS #1
    number = 0
    DO UNTIL EOF(1)
      LINE INPUT #1, temp$
      number = number + 1
    LOOP
    CLOSE #1
  END IF
END SUB

SUB LoadArrays (count, n$(), s$(), hw%(), ytd&())
  OPEN "PAYROLL.91" FOR INPUT AS #1
  PRINT "Lettura del file"
  FOR i = 1 TO count
    INPUT #1, n$(i), s$(i), hw%(i), ytd&(i)
  NEXT i
  CLOSE #1
END SUB

SUB SortByName (count, n$(), s$(), hw%(), ytd&())
  PRINT "Ordinamento"
  FOR i = 1 TO count
    swapped = 0
    FOR k = 1 TO count - i
      IF n$(k) > n$(k + 1) THEN   'Scambia gli elementi
        SWAP n$(k), n$(k + 1)     'dell'array quando questi
        SWAP s$(k), s$(k + 1)     'non sono ordinati
        SWAP hw%(k), hw%(k + 1)   'correttamente
        SWAP ytd&(k), ytd&(k + 1)
        swapped = 1
      END IF
    NEXT k
```

```
    'Se non sono state fatte delle modifiche nel ciclo
    'FOR k, significa che gli array sono ordinati
    IF swapped = 0 THEN EXIT FOR
  NEXT i
END SUB

SUB WriteArrays (count, n$(), s$(), hw%(), ytd&())
  PRINT "Scrittura del file ordinato"
  OPEN "PAYROLL.91" FOR OUTPUT AS #1
  FOR i = 1 TO count
    WRITE #1, n$(i), s$(i), hw%(i), ytd&(i)
  NEXT i
  CLOSE #1
END SUB
```

Nota: Se si dispone di un computer senza disco fisso, viene richiesto il dischetto contenente i file COMMAND.COM e SORT.EXE.

Si supponga che il file PAYROLL.91 sia molto lungo, che la società abbia assunto delle nuove persone e che i dati relativi ai nuovi impiegati siano contenuti nel file NEWEMP. Il Programma 7.8 riunisce questi due file in un solo file ordinato. Questa operazione viene effettuata copiando i record del file PAYROLL.91 in un altro file mentre ciascun record del file NEWEMP viene inserito nella posizione appropriata. Questo programma può essere modificato a seconda delle proprie esigenze per svolgere operazioni simili, come la cancellazione dei record relativi agli impiegati che hanno lasciato la società e la modifica del costo orario.

Programma 7.7 *Ordinamento del file PAYROLL.91 con il comando SORT del DOS*

```
REM Ordinamento del file PAYROLL.91 per nome   [7-7]
NAME "PAYROLL.91" AS "TEMPFILE"
SHELL "SORT <TEMPFILE >PAYROLL.91"
KILL "TEMPFILE"
END
```

Programma 7.8 *Unione di due file*

```
REM Aggiorna il file PAYROLL.91 aggiungendo nuovi nomi  [7-8]
REM Inserisce NEWEMP in PAYROLL.91
CALL OpenFiles
CLS
PRINT "Merging NEWEMP into PAYROLL.91"
nom1$ = ""     'Un valore nullo indica che deve essere letto il
               ' record successivo
nom2$ = ""
DO UNTIL (EOF(1) AND nom1$ = "") OR (EOF(2) AND nom2$ = "")
  'Legge il record successivo, se richiesto
```

```
    IF nom1$ = "" THEN
        INPUT #1, nom1$, ssn1$, hourlyWage1%, yearToDate1&
    END IF
    IF nom2$ = "" THEN
        INPUT #2, nom2$, ssn2$, hourlyWage2%, yearToDate2&
    END IF
    'Scrive il record cpn nome "minore" e indica di leggere
    'il record successivo
    IF nom1$ < nom2$ THEN
        WRITE #3, nom1$, ssn1$, hourlyWage1%, yearToDate1&
        nom1$ = ""
      ELSE  'nom2$ <= nom1$
        WRITE #3, nom2$, ssn2$, hourlyWage2%, yearToDate2&
        nom2$ = ""
    END IF
LOOP
'Tempfile o newemp sono stati elaborati completamente.
'Si devono ora scrivere gli altri dati dell'altro file.
IF nom1$ <> "" THEN
    WRITE #3, nom1$, ssn1$, hourlyWage1%, yearToDate1&
    DO UNTIL EOF(1)
      INPUT #1, nom1$, ssn1$, hourlyWage1%, yearToDate1&
      WRITE #3, nom1$, ssn1$, hourlyWage1%, yearToDate1&
    LOOP
  ELSE
    WRITE #3, nom2$, ssn2$, hourlyWage2%, yearToDate2&
    DO UNTIL EOF(2)
      INPUT #2, nom2$, ssn2$, hourlyWage2%, yearToDate2&
      WRITE #3, nom2$, ssn2$, hourlyWage2%, yearToDate2&
    LOOP
END IF
CLOSE
KILL "TEMPFILE"
PRINT "Unione completa"
END

SUB OpenFiles
  'Tempfile non deve esistere perché il file possa essere
  'rinominato, ma deve esistere perché possa essere
  'cancellato. Il comando OPEN seguente garantisce che tutto
  'funzioni correttamente..
  OPEN "TEMPFILE" FOR OUTPUT AS #1
  CLOSE #1
  KILL "TEMPFILE"
  NAME "PAYROLL.91" AS "TEMPFILE"
  OPEN "TEMPFILE" FOR INPUT AS #1
  OPEN "NEWEMP" FOR INPUT AS #2
  OPEN "PAYROLL.91" FOR OUTPUT AS #3
END SUB
```

CONSIDERAZIONI SUI FILE SEQUENZIALI

Si può utilizzare una variabile stringa per specificare il nome del file in un enunciato OPEN. Ciò risulta molto utile quando un programma deve elaborare alcuni file di dati differenti. L'utente, infatti, può specificare il nome del file da elaborare rispondendo a un enunciato INPUT.

Finora, gli ordinamenti dei file sono stati effettuati solo sull'unità disco di default, cioè quella da cui è stato avviato QBasic. Per lavorare con un file che si trova in un'altra unità disco, è sufficiente specificare la lettera di identificazione prima del nome del file. Per esempio, se il file PAYROLL.91 si trovasse in un dischetto nell'unità B, sarebbe sufficiente utilizzare l'enunciato

```
OPEN "B:PAYROLL.91" FOR INPUT AS #1
```

per accedere al file. Oltre a specificare l'unità disco, è possibile includere nel nome del file il percorso necessario per accedere alla sottodirectory desiderata.

Il numero massimo di file che possono essere aperti contemporaneamente dipende dall'impostazione del comando FILES del DOS nel file CONFIG.SYS. Se non viene utilizzato il comando FILES, viene utilizzata l'impostazione di default, 8. Si tenga presente che il DOS utilizza tre 'posizioni' e che altre possono essere usate da alcune periferiche. Per essere sicuri che QBasic possa aprire 15 file contemporaneamente, il comando FILES nel file CONFIG.SYS dovrebbe essere impostato almeno a 20.

I file sequenziali utilizzano lo spazio su disco in modo efficiente, risultano semplici da creare e gestire, ma comportano gli svantaggi seguenti:

- è spesso necessario leggere una porzione estesa del file per poter localizzare uno specifico elemento;
- non si può cancellare o modificare un singolo elemento del file. È necessario leggere tutte le informazioni dal file originale, modificare o cancellare l'elemento desiderato e scrivere i dati in un nuovo file.

Esiste un altro tipo di file, chiamato file ad accesso casuale, che non comporta gli svantaggi dei file sequenziali. Tuttavia, i file ad accesso casuale richiedono più spazio su disco e sono più difficili da gestire. Prima di esaminare i file ad accesso casuale, si introdurranno due nuovi tipi di variabile: stringhe a lunghezza fissa e record.

STRINGHE A LUNGHEZZA FISSA E RECORD

Normalmente, il tipo di dati di una variabile stringa viene dichiarato aggiungendo un segno dollaro al nome della variabile. L'assenza di un segno dollaro o l'uso dei suffissi

%, &, ! o #, dichiara la variabile come numerica. L'enunciato DIM fornisce un modo alternativo per la dichiarazione dei tipi di dati. Gli enunciati appropriati sono

```
DIM var AS STRING
DIM var AS INTEGER
DIM var AS LONG
DIM var AS SINGLE
DIM var AS DOUBLE
```

dove il nome della variabile viene scritto senza nessun simbolo di dichiarazione.

Programma 7.9 *Enunciato DIM per dichiarare dei tipi di variabile*

```
REM Enunciato DIM per dichiarare dei tipi di variabile [7-9]
CLS
DIM city AS STRING
city = "New York"
DIM pop AS SINGLE
pop = 7000000
PRINT city; pop
END

[Esecuzione]
New York 7000000
```

Il Programma 7.9 usa l'enunciato DIM per dichiarare dei tipi di variabile e assegna quindi dei valori. Anche se il programma funzionerebbe correttamente senza l'enunciato DIM pop AS SINGLE, si vedrà tra breve una situazione in cui risulta necessaria una dichiarazione di questo tipo.

Le variabili a lunghezza fissa non hanno il segno dollaro alla fine del nome, ma vengono specificate da un enunciato nella forma

```
DIM var AS STRING * n
```

dove *n* è un intero positivo. Dopo una dichiarazione di questo tipo, il valore di *var* sarà sempre una stringa di lunghezza *n*. Il valore iniziale è una stringa di *n* caratteri CHR$(0). Si supponga che *a$* sia una stringa normale e che venga impartito il comando

```
var = a$
```

Se *a$* è composta da più di *n* caratteri, alla variabile *var* vengono assegnati solo i primi *n* caratteri. Se *a$* contiene un numero di caratteri inferiore a *n*, vengono aggiunti degli spazi alla fine della stringa, in modo che *var* abbia lunghezza *n*.

Il Programma 7.10 utilizza delle variabili a lunghezza fissa. Dopo l'esecuzione, si può notare che San Francisco è stata troncato al nono carattere e che a Detroit sono stati aggiunti due spazi.

Bisogna prestare molta attenzione quando si confronta una stringa a lunghezza variabile con una stringa a lunghezza fissa, o quando si confrontano due variabili a lunghezza fissa composte da un numero diverso di caratteri. Nel Programma 7.11, le stringhe assegnate a *city, town$* e *municipality* hanno, rispettivamente, una lunghezza pari a 9, 7 e 12, e sono quindi differenti.

Programma 7.10 *Uso del comando DIM per dichiarare delle variabili a lunghezza fissa*

```
REM Uso di variabili a lunghezza fissa  [7-10]
CLS
PRINT "123456789"
DIM city AS STRING * 9
city = "San Francisco"
PRINT city
city = "Detroit"
PRINT city; "MI"
PRINT LEN(city)
END

[Esecuzione]
123456789
San Franc
Detroit  MI
 9
```

Ci sono dei casi in cui risulta necessario trascurare la presenza degli spazi aggiunti a una variabile a lunghezza fissa, come nel caso di *city* nel Programma 7.11. A questo scopo, si può utilizzare la funzione RTRIM$. Se *a$* fosse una stringa normale o a lunghezza fissa, il valore di

```
RTRIM$(a$)
```

corrisponderebbe al contenuto di *a$* senza gli spazi in coda. Ad esempio, la funzione RTRIM$("ciao   ") restituisce la stringa "ciao". Se nel Programma 7.11 si cambiasse il blocco IF in

```
IF (RTRIM$(city)=town$) AND (RTRIM$(city)=RTRIM$(municipality)) THEN
   PRINT "uguali"
ELSE
   PRINT "diverse"
END IF
```

la prima riga dell'output sarebbe "uguali".

Un array di lunghezza fissa viene dichiarato da un enunciato nella forma

```
DIM nomeArray(a TO b) AS STRING * n
```

Quando si passano delle stringhe a lunghezza fissa a una procedura, il parametro nell'enunciato SUB o FUNCTION deve essere una stringa a lunghezza variabile.

Programma 7.11 *Differenza tra le stringhe normali e quelle a lunghezza fissa*

```
REM Variabili a lunghezza fissa e stringhe normali [7-11]
CLS
DIM city AS STRING * 9
DIM municipality AS STRING * 12
town$ = "Chicago"
city = "Chicago"
municipality = "Chicago"
IF (city = town$) OR (city = municipality) THEN
    PRINT "uguali"
  ELSE
    PRINT "diverse"
END IF
PRINT "123456789012345"
PRINT city + "***"
PRINT town$ + "***"
PRINT municipality + "***"
END

[Esecuzione]
diverse
123456789012345
Chicago  ***
Chicago***
Chicago     ***
```

RECORD

Finora sono stati esaminati quattro tipi di variabili: numeriche, stringa, array e stringhe a lunghezza fissa. Le stringhe e i numeri sono dei tipi di dati 'incorporati' che possono venire usati senza essere dichiarati. Per contro, gli array e le stringhe a lunghezza fissa sono tipi di dati definiti dall'utente che devono essere dichiarati con l'enunciato DIM prima di essere utilizzati.

Nota: L'unica eccezione sono gli array il cui indice è compreso tra 0 e 10.

Un record è un tipo di dati definito dall'utente che comprende un gruppo di variabili tra loro relazionate di tipo differente.

La Figura 7.5 mostra una scheda che può essere utilizzata per contenere dei dati relativi ai college. Le tre parti che compongono i dati, il nome, lo stato e l'anno di fondazione, sono chiamate *campi*. La lunghezza di un campo corrisponde al numero di spazi loro riservati. I tre campi della scheda hanno, rispettivamente, una lunghezza pari a 30, 2 e 4. Ogni campo è in realtà una variabile in cui si possono memorizzare delle informazioni. L'organizzazione della scheda può essere identificata da un nome, ad esempio CollegeData, a cui ci si riferisce come *tipo di record*.

Per scopi di programmazione, l'organizzazione del record viene dichiarata dal blocco di enunciati

```
TYPE CollegeData
   nom AS STRING * 30
   state AS STRING * 2
   yearFounded AS SINGLE
END TYPE
```

Un record in grado di contenere i dati relativi a un college specifico viene dichiarato tramite l'enunciato

```
DIM college AS CollegeData
```

Questo enunciato crea le tre variabili *college.nom*, *college.state* e *college.yearFounded*.

In generale, un tipo di record viene creato da un blocco TYPE della forma

```
TYPE TipoRecord
   nomeCampo AS tipoCampo
   nomeCampo AS tipoCampo
   .
END TYPE
```

e un record viene dichiarato da un enunciato della forma

```
DIM nomeRecord AS TipoRecord
```

dove *TipoRecord* è il nome del tipo di dati definito dall'utente, *nomeCampo* è il nome di uno dei campi del record, e *tipoCampo* è STRING * *n*, oppure uno dei tipi numerici: INTEGER, SINGLE, LONG o DOUBLE.

Name: _ Stato: _ Anno di Fondazione: _

Figura 7.5 *Una scheda con tre campi*

Il Programma 7.12 mostra l'uso dei record

Programma 7.12 *Uso dei record*

```
REM Uso dei record  [7-12]
CLS
TYPE CollegeDATA
  nom AS STRING * 30
  state AS STRING * 2
  yearFounded AS SINGLE
END TYPE
DIM college AS CollegeDATA
INPUT "Nome"; college.nom
INPUT "Stato"; college.state
INPUT "Anno di fondazione"; college.yearFounded
century = 1 + INT(college.yearFounded / 100)
PRINT RTRIM$(college.nom); " è stata fondata nel"; century;
PRINT "secolo in "; college.state
END

[Esecuzione]
Nome? Boston University
Stato? MA
Anno di fondazione? 1839
Boston University è stata fondata nel 19 secolo in MA
```

Gli enunciati TYPE possono apparire solamente nella porzione principale di un programma e mai nelle funzioni o nei sottoprogrammi. Tuttavia, gli enunciati DIM possono essere usati nelle funzioni e nei sottoprogrammi per dichiarare una variabile come record. Quando si passano dei record a un sottoprogramma o a una funzione, il parametro nell'enunciato SUB o FUNCTION deve avere la forma

```
parametro AS tiporecord
```

Il Programma 7.13 usa dei sottoprogrammi per eseguire le stesse operazioni del Programma 7.12.

I record sono simili agli array nel senso che entrambi memorizzano ed accedono ai dati usando un nome comune. Tuttavia, gli elementi in un array devono essere dello stesso tipo, mentre i campi di un record possono comprendere tipi di dati differenti. Inoltre, i diversi elementi di un array vengono identificati dagli indici, mentre i campi di un record sono identificati dal nome che segue il punto.

Ogni carattere di una stringa viene memorizzato in una locazione di memoria chiamata byte. Quindi, un campo di tipo STRING * *n* richiede *n* byte di memoria. I numeri interi, interi lunghi, a precisione singola e a precisione doppia vengono memorizzati, rispettivamente, in 2, 4, 4 e 8 byte. Se *recVar* fosse un record definito

dall'utente, il valore di LEN(*recVar*) corrisponderebbe alla somma del numero di byte usati dai campi di *recVar*.

Programma 7.13 *Passaggio di record ai sottoprogrammi*

```
REM Uso dei record con le procedure  [7-13]
CLS
TYPE CollegeData
  nom AS STRING * 30
  state AS STRING * 2
  yearFounded AS SINGLE
END TYPE
DIM college AS CollegeData
CALL GetData(college)
CALL DisplayStatement(college)
END

SUB DisplayStatement (school AS CollegeData)
  REM Visualizza il nome, il secolo e lo stato
  century = 1 + INT(school.yearFounded / 100)
  PRINT RTRIM$(school.nom); " è stata fondata nel"; century;
  PRINT "secolo in "; school.state
END SUB

SUB GetData (school AS CollegeData)
  REM Richiede il nome, il secolo e lo stato
  INPUT "Nome"; school.nom
  INPUT "Stato"; school.state
  INPUT "Anno di fondazione"; school.yearFounded
END SUB
```

Oltre a essere dichiarati come tipi di dati numerici o stringhe a lunghezza fissa, gli elementi di un record definito dall'utente possono essere dichiarati come altri tipi di record. Il Programma 7.14 si serve di questa caratteristica.

Programma 7.14 *Un record definito dall'utente con un record come elemento*

```
REM Uso di record con dei record come elementi  [7-14]
TYPE NameType
  first AS STRING * 15
  last AS STRING * 15
END TYPE
TYPE AddressType
  street AS STRING * 30
  city AS STRING * 15
  state AS STRING * 2
  zipCode AS STRING * 5
END TYPE
```

```
TYPE PersonData
  nom AS NameType
  address AS AddressType
END TYPE
DIM president AS PersonData
CLS
CALL GetData(president)
CALL DisplayData(president)
END

SUB DisplayData (person AS PersonData)
  PRINT RTRIM$(person.nom.first); " ";
  PRINT RTRIM$(person.nom.last)
  PRINT RTRIM$(person.address.street)
  PRINT RTRIM$(person.address.city); ", ";
  PRINT person.address.state; " "; person.address.zipCode

END SUB

SUB GetData (person AS PersonData)
  REM Request the name, address, and phone number
  INPUT "Nome: ", person.nom.first
  INPUT "Cognome: ", person.nom.last
  INPUT "Indirizzo: ", person.address.street
  INPUT "Città: ", person.address.city
  INPUT "Probincia: ", person.address.state
  INPUT "CAP: ", person.address.zipCode
END SUB

[Esecuzione]
Nome: Giorgio
Cognome: Cespuglio
Indirizzo: Via Lunarca, 7
Città: Mazzo di Rho
Provincia: MI
CAP: 20230

Giorgio Cespuglio
Via Lunarca, 7
Mazzo di Rho, MI 20230
```

FILE AD ACCESSO CASUALE

I dati in un file ad accesso casuale possono essere paragonati alle informazioni contenute in un registro contabile con le righe numerate. Ogni riga può essere letta e scritta senza prima esaminare gli altri dati del registro. Nella Figura 7.6, ogni riga è divisa in quattro parti. Ciascuna riga costituisce un record e ogni porzione della riga

un campo. Dei quattro campi mostrati in Figura 7.6, due contengono dei dati stringa e gli altri due dei dati numerici (i dati numerici sono stati codificati in stringhe di due e quattro caratteri). Sommando gli spazi riservati per ciascun campo, si può ottenere il massimo numero di caratteri memorizzabile in un record (27+11+2+4=44). In questo caso si può affermare che ogni record è lungo 44 caratteri.

Un record definito dall'utente viene usato per leggere e scrivere i record di un file ad accesso casuale. I campi del record definito dall'utente devono essere uguali ai campi dei record del file ad accesso casuale. Per esempio, considerando il file della Figura 7.6, l'enunciato TYPE appropriato è

```
TYPE EmployeeType
   nom AS STRING * 27
   ssn AS STRING * 11
   hourlyWage AS INTEGER
   yearToDate AS LONG
END TYPE
```

e l'enunciato DIM appropriato è

```
DIM employee AS EmployeeType
```

dove *employee* è un record definito dall'utente di tipo EployeeType.

Nel caso dei file ad accesso casuale, è sufficiente un solo enunciato per aprire il file per qualsiasi scopo: creazione, scrittura, modifica o lettura. Si supponga che *n* sia il numero di riferimento scelto per il file. L'enunciato

```
OPEN nomefile FOR RANDOM AS #n LEN = LEN(recVar)
```

consente di scrivere, leggere, modificare o aggiungere dei record al file specificato. La funzione LEN(*recVar*) restituisce la lunghezza del record.

Figura 7.6 *Un registro contabile*

Si supponga di aver aperto un file ad accesso casuale e di aver impartito un enunciato TYPE. A questo punto, per inserire dei dati in un record, si proceda nel modo seguente:

1. assegnare un valore a ciascun campo del record definito dall'utente;
2. inserire i dati nel record *r* del file il cui numero di riferimento è uguale a *n* tramite l'enunciato:

```
PUT #n, r, recVar
```

Il Programma 7.15 genera il file ad accesso casuale della Figura 7.6. Si noti che il ciclo all'interno del programma viene controllato da un valore sentinella anche se il numero di record può essere facilmente determinato. Grazie a questa tecnica non è necessario apportare delle modifiche alla porzione principale del programma se vengono aggiunti dei nuovi enunciati alla fine del programma stesso. Si supponga di aver dichiarato il record *recVar* e di aver aperto un file ad accesso casuale con numero di riferimento *n*. L'enunciato

```
GET #n, r, recVar
```

assegna il record *r* del file con numero di riferimento *n* a *recVar*.

Programma 7.15 *Creazione del file ad accesso casuale PAYROLL.91*

```
REM Creazione del file as accesso casuale PAYROLL.91R  [7-15]
TYPE EmployeeType
  nom AS STRING * 27
  ssn AS STRING * 11
  hourlyWage AS INTEGER  'costo orario
  yearToDate AS LONG     'importo dovuto
END TYPE
DIM employee AS EmployeeType
OPEN "PAYROLL.91R" FOR RANDOM AS #1 LEN = LEN(employee)
recordNumber = 1
READ employee.nom, employee.ssn, employee.hourlyWage,
employee.yearToDate
DO UNTIL RTRIM$(employee.nom) = "EOD"
  PUT #1, recordNumber, employee
  recordNumber = recordNumber + 1
  READ employee.nom, employee.ssn
  READ employee.hourlyWage, employee.yearToDate
LOOP
CLOSE #1
REM -- Dati:nome, numero, costo orario, importo dovuto
DATA "Mario,Rossi", 123-45-6789, 17000, 19091000
DATA "Alberto,Rubinetti", 456-98-7654, 15000, 1537650
DATA "Davide,Villa", 238-91-2355, 25000, 934700
DATA EOD,"",0,0
```

Il numero totale di caratteri nel file con numero di riferimento *n* viene fornito dalla funzione

```
LOF(n)
```

La divisione del valore della funzione LOF per la lunghezza del record, LEN(*recVar*), restituisce il numero di record presenti nel file ad accesso casuale. Il Programma 7.16 usa questa tecnica per visualizzare l'intero contenuto del file PAYROLL.91.

Programma 7.16 *Visualizzazione del file PAYROLL.91*

```
REM Prelevamento dei record dal file PAYROLL.91R  [7-16]
DEFINT I
TYPE EmployeeType
  nom AS STRING * 27
  ssn AS STRING * 11
  hourlyWage AS INTEGER  'costo orario
  yearToDate AS LONG     'importo dovuto
END TYPE
DIM employee AS EmployeeType
OPEN "PAYROLL.91R" FOR RANDOM AS #1 LEN = LEN(employee)
CLS
PRINT "                    Numero"
PRINT "                    di identi-     Costo    Importo"
PRINT "Nome                ficazione      Orario   Dovuto"
PRINT "------------------  -----------    ------   ---------"
fm$ = "\                \  \         \    #####    ########"
FOR i = 1 TO LOF(1) / LEN(employee)
  GET #1, i, employee
  PRINT USING fm$; employee.nom; employee.ssn; employee.hourlyWage /_
              100; employee.yearToDate / 100
NEXT i
CLOSE #1
END
```

```
[Esecuzione]
                    Numero"
                    di identi-     Costo    Importo
Nome                ficazione      Orario   Dovuto
------------------  -----------    ------   ---------
Mario,Rossi         123-45-6789    17000    19091000
Alberto,Rubinetti   456-98-7654    15000     1537650
Davide,Villa        238-91-2355    25000      934700
```

CONSIDERAZIONI SUI FILE AD ACCESSO CASUALE

1. I file ad accesso casuale vengono anche denominati *file ad accesso diretto*. Dato che ogni record ha lo stesso numero di caratteri, il computer può facilmente trovare la posizione di un record specifico senza doverlo cercare;
2. a differenza dei file sequenziali, i file ad accesso casuale non devono necessariamente essere chiusi tra una sessione di scrittura e una di lettura;
3. i record non devono essere riempiti in ordine sequenziale. Per esempio, dopo aver aperto un file, si potrebbe impartire il comando PUT #*n*, 9, *recVar*;
4. un campo di un record definito dall'utente conserva il suo valore a meno che questo non venga modificato da un comando GET o LET;
5. se il numero di record *r* viene omesso in un enunciato GET o PUT, il numero di record utilizzato sarà quello seguente al numero usato più di recente in un enunciato GET o PUT (o il primo record se non sono mai stati impartiti dei comandi GET o PUT). Ad esempio, se si aggiungesse l'enunciato PUT #1, *recVar* al Programma 7.15 esattamente prima dell'istruzione che chiude il file, le informazioni relative all'impiegato Davide Villa verrebbero duplicate nel record 4 del file PAYROLL.91R;
6. molti utenti inseriscono dei record in un file ad accesso casuale senza tenere traccia dei relativi numeri. Se è aperto il file numero *n*, la funzione

   ```
   LOC(n)
   ```

   restituisce il numero del record elaborato dall'ultimo comando GET o PUT;
7. un metodo alternativo per leggere e scrivere dei dati in un file ad accesso casuale è costituito dal cosiddetto metodo del buffer (questo metodo è l'unico disponibile nel BASIC standard e nelle versioni di QuickBASIC precedenti alla 4.0). Benché il metodo del buffer richieda un maggiore sforzo di programmazione, è l'unico modo per scrivere un programma di database quando varia la struttura del file. Con questo metodo viene usato un enunciato FIELD per assegnare a ciascun campo un nome e una larghezza. Le informazioni vengono quindi inserite nel file tramite una doppia procedura. Un porzione di memoria, denominata buffer, viene automaticamente riservata per il file. I dati relativi a un singolo record vengono inseriti nel buffer, un campo alla volta. Questa operazione viene eseguita dai comandi LSET e RSET. Infine, l'intero record viene copiato nel file come record numero *r* mediante l'enunciato

   ```
   PUT #n, r
   ```

   Per leggere dei record è sufficiente seguire la procedura inversa. L'enunciato

   ```
   GET #n, r
   ```

carica una copia del record numero *r* nel buffer. Si può quindi accedere ai singoli campi facendo riferimento al loro nome. Si tenga presente che i dati numerici devono essere convertiti in stringhe prima di essere inseriti nel file. Le funzioni MKI$, MKL$, MKS$ e MKD$ consentono di convertire un dato numerico in stringa, mentre le funzioni CVI, CVL, CVS e CVD svolgono l'operazione inversa. Il Programma 7.17 è una versione del Programma 7.16 che si serve del metodo del buffer.

Programma 7.17 *Il metodo del buffer con un file ad accesso casuale*

```
REM Prelevamento dei dati dal file PAYROLL.91R   [7-17]
OPEN "PAYROLL.91R" FOR RANDOM AS #1 LEN = 44
FIELD #1, 27 AS nom$, 11 AS ssn$, 2 AS hrWg$, 4 AS yrToDt$
CLS
PRINT "                           Numero"
PRINT "                           di identi-      Costo    Importo"
PRINT "Nome                       ficazione       Orario   Dovuto"
PRINT "-------------------        -----------     ------   --------"
f$ = "\                       \  \         \     #####    ########"
FOR i = 1 TO LOF(1) \ 44
  GET #1, i
  PRINT USING f$; nom$; ssn$; CVI(hrWg$) / 100; CVL(yrToDt$) / 100
NEXT i
CLOSE #1
END
```

FILE BINARI

Un file binario è il tipo di file più rudimentale. Esso offre una grande flessibilità, ma richiede un notevole sforzo da parte del programmatore. Si può pensare a un file binario come a una lunga sequenza di caratteri *senza struttura.* I caratteri occupano le posizioni 1, 2, 3 e così via. Si può saltare in qualsiasi posizione e leggere e scrivere un qualsiasi numero di caratteri. Quando si inseriscono dei nuovi caratteri in un file binario, QBasic sovrascrive gli eventuali caratteri presenti nelle posizioni in cui avviene l'inserimento.

Analogamente ai file ad accesso casuale, i file binari dispongono di un solo enunciato OPEN che serve per tutti gli scopi. L'enunciato

```
OPEN nomefile FOR BINARY AS #n
```

assegna il numero di riferimento *n* al file binario specificato che può, a questo punto, essere letto e modificato. In qualsiasi momento esiste una locazione nel file denominata *posizione corrente del file.* Inizialmente, questa posizione è uguale a 1. Le operazioni

principali per accedere a un file binario vengono svolte dagli enunciati SEEK, PUT e GET, e dalle funzioni SEEK e LOF.

Se *n* è il numero di riferimento di un file binario, l'enunciato

```
SEEK #n, r
```

cambia la posizione corrente del file in *r*. L'enunciato

```
PUT #n, r, var
```

inserisce il valore di *var* nel file, partendo dalla locazione *r* e spostando la posizione corrente del file a una distanza pari alla lunghezza di *var*. L'enunciato

```
GET #n, r, var
```

inizia la lettura dalla locazione *r*, legge un numero di byte pari alla lunghezza di *var*, li assegna a *var* e sposta la posizione corrente del file a una distanza uguale alla lunghezza di *var*.

In qualsiasi momento, la funzione

```
SEEK(n)
```

restituisce la posizione corrente del puntatore nel file binario il cui numero di riferimento è uguale a *n*, mentre la funzione

```
LOF(n)
```

fornisce la lunghezza del file. Il Programma 7.18 mostra alcuni esempi dell'uso di queste funzioni. I commenti inseriti dopo gli enunciati PUT mostrano il contenuto del file in quel determinato momento.

Si può aprire un qualsiasi file come binario e utilizzare i comandi SEEK, PUT e GET. Ad esempio, un file sequenziale può essere aperto come binario e può essere modificato senza dover riscrivere tutti i dati in esso contenuti. A titolo di esempio, il Programma 7.19 modifica il file sequenziale PAYROLL.91 riportato in Figura 7.1. Si noti, nella Figura 7.1, che i dati relativi al costo orario dell'impiegato Mario Rossi iniziano con il ventinovesimo carattere del file.

I file creati fuori dall'ambiente QBasic possono avere dei formati particolari. Ad esempio, i file creati dai programmi di foglio elettronico e da quelli di database hanno un loro formato specifico e possono essere aperti ed esaminati solo come file binari. Il Programma 7.20 utilizza la modalità binaria per consentire la visualizzazione di un qualsiasi tipo di file. Ogni riga visualizzata consiste di due parti: la porzione a sinistra riporta i valori ASCII di 15 caratteri letti dal file, mentre la parte di destra mostra i

caratteri corrispondenti ai codici ASCII, utilizzando un punto per rappresentare i simboli che non fanno parte dei 96 caratteri di tastiera standard.

Programma 7.18 *Uso di un file binario*

```
REM Uso di un file binario  [7-18]
OPEN "DEMOFILE" FOR BINARY AS #1
abcde$ = "abcde"
fgh$ = "fgh"
FGHIJ$ = "FGHIJ"
b$ = "  "               'LEN(b$) = 2
a$ = "A"
PUT #1, 1, abcde$       'abcde
PUT #1, 6, fgh$         'abcdefgh
SEEK #1, 3              'Si sposta nella posizione 3
GET #1, , b$            'Legge 2 caratteri; il valore di b$ è "cd"
PRINT SEEK(1)           'Mostra la posizione corrente, 5
PUT #1, , FGHIJ$        'abcdFGHIJ
SEEK #1, 1              'Si sposta all'inizio del file
PUT #1, , a$            'AbcdFGHIJ
SEEK #1, LOF(1)         'Si sposta alla fine del file
b$ = "kl"
done$ = "fatto"
PUT #1, , b$            'AbcdFGHIkl
SEEK #1, SEEK(1) - 5 'Si sposta di 5 posizioni a sinistra
PUT #1, , done$         'AbcdFdonel
CLOSE #1                'Chiude il file
END

[Esecuzione]
 5
```

Programma 7.19 *Apertura di un file sequenziale come file binario*

```
REM Modifica il costo orario di Mario Rossi [7-19]
OPEN "PAYROLL.91" FOR BINARY AS #1
SEEK #1, 29            'sposta il puntatore sul 29° carattere
pay$ = "18500"
PUT #1, , pay$     'cambia dal carattere 29 al 33
CLOSE #1
END
```

Il Programma 7.21 non solo consente di visualizzare un qualsiasi tipo di file, ma permette di modificarlo. I caratteri vengono visualizzati verticalmente uno per riga insieme alla loro posizione nel file. Si possono visualizzare e modificare al massimo 19 caratteri per volta.

Programma 7.20 *Visualizzazione di un qualsiasi tipo di file*

```
REM Decodifica un file qualsiasi in ASCII  [7-20]
DIM byte AS STRING * 1
CLS
INPUT "File di origine: ", filename$
OPEN filename$ FOR BINARY AS #1
text$ = ""
FOR index = 1 TO LOF(1)     'elabora l'intero file
  'Legge un singolo carattere dal file
  GET #1, index, byte
  'Visualizza il valore ASCII del carattere
  PRINT USING "\  \"; STR$(ASC(byte));
  'Se il carattere non è standard lo cambia in un punto
  IF (ASC(byte) < 32) OR (ASC(byte) > 127) THEN byte = "."
  'Aggiunge il carattere in byte$ nella stringa text$
  text$ = text$ + byte
  'Dopo la visualizzazione di 15 valori ASCII, mostra il
  'testo corrispondente
  IF index MOD 15 = 0 THEN
    PRINT "  "; text$
    text$ = ""
  END IF
NEXT index
PRINT TAB(63); text$   'Visualizza l'ultima riga
CLOSE #1
END
```

Il sottoprogramma DisplayCommands mostra nella parte inferiore dello schermo i comando disponibili nell'editor. Per evitare che questi comandi vengano cancellati, tutti i dati visualizzati sono confinati nelle prime 19 righe dello schermo.

Il sottoprogramma AdvanceToNextByte usa la funzione GET per leggere il carattere nella posizione corrente dopo aver verificato che non sia stata raggiunta la fine del file. Questo carattere viene quindi visualizzato sullo schermo. Dato che la funzione GET sposta automaticamente la posizione corrente del file di una locazione, qualsiasi azione da svolgere sul carattere appena visualizzato deve iniziare con l'enunciato SEEK, al fine di spostare la posizione corrente del file indietro di una locazione.

Programma 7.21 *Un editor*

```
REM Esamina e modifica i file byte per byte  [7-21]
DIM SHARED byte AS STRING * 1
DEFINT A-Z
bottomLine = 19
CLS
INPUT "File di origine: ", filename$
OPEN filename$ FOR BINARY AS #1
CALL DisplayCommands
currentLine = 1
GET #1, 1, byte
CALL Display(byte)
CALL ReadIn(action$)
DO UNTIL action$ = "Q"
  SELECT CASE action$
    CASE "N", CHR$(13)
      CALL AdvanceToNextByte
    CASE "R"
      CALL ReplaceCurrentByte
    CASE "G"
      CALL GoToByteRelativeToStart
    CASE "J"
      CALL JumpToByteFromHere
  END SELECT
  CALL ReadIn(action$)
LOOP
CLOSE #1
END

SUB AdvanceToNextByte
  SHARED currentLine
  IF NOT EOF(1) THEN              'Se non è alla fine del file
    GET #1, , byte                'avanza al byte successivo
    CALL Display(byte)
  END IF
END SUB

SUB Display (byte AS STRING)
  SHARED currentLine, bottomLine
  LOCATE currentLine, 1
  'Mostra la posizione del byte nel file e il valore ASCII
  PRINT USING "##########  \   \"; SEEK(1) - 1; STR$(ASC(byte));
  'Visualizza la descrizione del byte
  SELECT CASE byte
    CASE CHR$(0) TO CHR$(31)
      PRINT "^"; CHR$(ASC(byte) + 64);
    CASE CHR$(32) TO CHR$(127)
      PRINT byte;
```

```
    CASE CHR$(128) TO CHR$(159)
      PRINT "@^"; CHR$(ASC(byte) - 64);
    CASE CHR$(160) TO CHR$(255)
      PRINT "@"; CHR$(ASC(byte) - 128);
  END SELECT
  PRINT "        ";
  currentLine = (currentLine MOD bottomLine) + 1
  LOCATE currentLine, 1: PRINT SPACE$(30);
  LOCATE currentLine, 1
END SUB

SUB DisplayCommands
  SHARED bottomLine
  CLS
  LOCATE bottomLine + 2, 1
  PRINT "N o [INVIO]  = passa al byte successivo"
  PRINT "J            = si sposta in avanti o indietro"
  PRINT "G            = va a un byte specifico"
  PRINT "R            = sostituisce questo byte"
  PRINT "Q            = salva il file ed esce";
END SUB

SUB GoToByteRelativeToStart
  SHARED currentLine
  INPUT "distanza --> ", size
  IF (size >= 1) AND (size <= LOF(1)) THEN
      SEEK #1, size            'cambia la posizione corrente
      GET #1, , byte           'in quella definita dall'utente
      CALL Display(byte)       'e visualizza il byte letto
    ELSE
      CALL ReportError
  END IF
END SUB

SUB JumpToByteFromHere
  SHARED currentLine
  INPUT "salto in byte --> ", size
  IF (size + SEEK(1) >= 1) AND (size + SEEK(1) <= LOF(1)) THEN
      SEEK #1, size + SEEK(1)         'cambia la posizione corrente
      GET #1, , byte                  'in base a LOC(1) che è
      CALL Display(byte)              'la posizione dell'ultimo
                                      'byte letto
    ELSE
      CALL ReportError
  END IF
END SUB

SUB ReadIn (action$)
  SHARED currentLine
```

```
    PRINT "azione?";              'mostra il prompt, legge un tasto
    action$ = UCASE$(INPUT$(1))   'e lo converte in maiuscolo
    LOCATE currentLine, 1: PRINT "        "; 'cancella il prompt
    LOCATE currentLine, 1
  END SUB

  SUB ReplaceCurrentByte
    SHARED currentLine, bottomLine
    currentLine = currentLine - 1    'sposta il cursore in alto
    IF currentLine = 0 THEN
        currentLine = bottomLine     'passa in fondo allo schermo
    END IF
    LOCATE currentLine, 1            'inserisce il cursore
    PRINT "sostituire con -->";      'visualizza il prompt
    text$ = INPUT$(1)                'legge il carattere
    SEEK #1, SEEK(1) - 1             'Annulla l'avanzamento
                                     'causato dall'ultimo GET
    PUT #1, , text$                  'sostituisce il byte
    CALL Display(text$)
  END SUB

  SUB ReportError
    SHARED currentLine
    'Mostra un messaggio di errore sulla riga successiva
    LOCATE currentLine + 1, 1: PRINT "Non è nel file";
    SLEEP 1
    LOCATE currentLine + 1, 1: PRINT SPACE$(30);
    LOCATE currentLine, 1: PRINT SPACE$(30);
    LOCATE currentLine, 1
  END SUB
```

Il sottoprogramma ReplaceCurrentByte riposiziona il cursore sulla riga contenente l'ultimo carattere visualizzato e richiede il carattere di sostituzione. Una volta specificato un nuovo carattere, viene usato l'enunciato SEEK per spostare il puntatore del file indietro di una posizione, cioè sul carattere precedentemente letto con GET. Infine, il comando PUT effettua la sostituzione.

Il sottoprogramma GoToByteRelativeToStart richiede all'utente di specificare la nuova posizione corrente del puntatore. Questa posizione è relativa all'inizio del file. Il menu dei comandi mostra l'intervallo di valori disponibile compreso tra 1, l'inizio del file, e LOF(1), la fine del file. Se l'utente specifica una posizione valida, viene impartito il comando SEEK per raggiungere la nuova locazione e visualizzare il carattere ivi contenuto.

Il sottoprogramma JumpToByteFromHere richiede all'utente di specificare il valore da usare per lo spostamento del puntatore del file partendo dalla posizione corrente. Un valore negativo causa uno spostamento all'indietro, verso l'inizio del file, mentre un valore positivo sposta il puntatore in avanti. L'utente viene avvertito nel caso in cui

la nuova posizione non si trovi nel file. In caso contrario, viene usato il comando SEEK per cambiare la posizione corrente del file e visualizzare il nuovo carattere raggiunto.

Il sottoprogramma ReadIn chiede all'utente di impartire uno dei cinque comandi visualizzati nella parte inferiore dello schermo. Una volta premuto un tasto, la richiesta viene cancellata.

Il sottoprogramma DisplayByte visualizza la posizione corrente del file, il codice ASCII corrispondente e una descrizione del carattere, e cancella la 'riga successiva' visualizzata sullo schermo. Se la riga corrente è la 19, la riga successiva è la 1; in caso contrario, la riga successiva è quella che segue la riga corrente. La descrizione visualizzata per un carattere dipende dal suo codice ASCII. I 96 caratteri standard di tastiera (valore ASCII compreso tra 32 e 127) vengono visualizzati come tali, mentre i caratteri di controllo (valore ASCII compreso tra 0 e 31) vengono rappresentati con una cuspide (^) seguita dal carattere il cui codice ASCII è uguale al codice del carattere di controllo sommato a 64. Ad esempio, il carattere con codice ASCII 7 viene visualizzato come ^G (il codice ASCII della lettera G è uguale a 71). DisplayByte visualizza i caratteri del codice esteso (quelli compresi tra 128 e 255) con il simbolo @ seguito dalla descrizione del carattere il cui codice ASCII è uguale al codice esteso del simbolo meno 128. Per esempio, il carattere con codice ASCII 135 viene rappresentato come @^G, mentre il carattere con codice 193 come @A (il codice ASCII della lettera A è 65).

Il programma utilizza il sottoprogramma ReportError per indicare all'utente che un comando di spostamento ha generato una posizione che non esiste nel file. ReportError visualizza un messaggio di errore, attende un secondo e cancella quindi il messaggio.

In questo capitolo si è visto come creare e modificare dei file, e come esaminare dei file creati da altri programmi. Nel prossimo capitolo si cambierà argomento e si esamineranno i comandi relativi alla gestione della grafica e del suono.

CAPITOLO 8

GRAFICA E SUONI

QBasic dispone di numerosi comandi che consentono di gestire la grafica e il suono. Questo capitolo spiega come creare e animare delle immagini a colori e in bianco e nero, e come aggiungere nei programmi degli effetti sonori. Il Capitolo 9 spiegherà come ottenere delle rappresentazioni grafiche delle funzioni e il Capitolo 10 come rappresentare graficamente i propri dati.

GRAFICI

I grafici possono essere generati se si dispone di una scheda grafica o di un video monocromatico collegato a un adattatore Hercules. Se si utilizza una scheda Hercules, è necessario eseguire il programma MSHERC.COM dal DOS prima di avviare QBasic. Tutti gli esempi riportati in questo capitolo presumono che si disponga di un adattatore video grafico.

I monitor che si possono usare con un PC IBM o compatibile possono essere di quattro tipi: RGB, ad alta risoluzione, monocromatici e compositi. Un monitor può essere collegato a una scheda CGA (Color Graphics Adapter), EGA (Enhanced Graphics Adapter) o VGA (Video Graphics Adapter).

La configurazione più avanzata è costituita dalla scheda VGA collegata a un monitor ad alta risoluzione. Questo capitolo illustra inizialmente le tecniche che funzionano con qualsiasi tipo di configurazione, e si sofferma quindi sulle capacità dei sistemi EGA e VGA.

MODALITÀ GRAFICHE

Le due modalità grafiche disponibili in tutti i sistemi grafici sono le modalità a media risoluzione e ad alta risoluzione. La grafica a media risoluzione consente di utilizzare fino a quattro colori contemporaneamente, mentre quella ad alta risoluzione solo il bianco e il nero.

I PUNTI SULLO SCHERMO

Lo schermo grafico è composto da una griglia suddivisa in piccoli rettangoli chiamati punti o pixel (pixel sta per **pi**cture **el**ement). In entrambe le modalità grafiche ci sono 200 pixel verticali; in orizzontale, invece, ci sono 320 pixel in media risoluzione e 640 pixel in alta risoluzione (si veda la Figura 8.1).

Nota: Se si utilizza un monitor monocromatico collegato a una scheda Hercules ci sono 348 pixel in verticale e 720 in orizzontale.

Ciascun pixel viene identificato da una coppia di numeri, chiamati coordinate. Le coordinate del punto nell'angolo in alto a sinistra dello schermo sono (0,0). Si può raggiungere un qualsiasi punto della griglia specificando le coordinate (x, y) sulla base del punto in alto a sinistra, e spostandosi x punti verso destra e y punti verso il basso, come mostrato in Figura 8.2. Per esempio, il centro dello schermo ha coordinate (160, 100) in media risoluzione, e (320, 100) in alta risoluzione.

Nota: Se si utilizza un monitor monocromatico collegato a una scheda Hercules, il centro dello schermo ha coordinate (360, 174).

Gli enunciati grafici principali disponibili in QBasic sono SCREEN, PSET, PRESET, LINE, CIRCLE, DRAW, PAINT, COLOR, GET, PUT, POINT, VIEW, WINDOW e PMAP. Il comando SCREEN consente di specificare una modalità grafica. Per visualizzare dei punti, delle linee e dei cerchi, si possono usare, rispettivamente, i comandi PSET, LINE e CIRCLE. L'enunciato DRAW genera delle figure sullo schermo, mentre l'istruzione COLOR, in media risoluzione, consente di selezionare i colori da utilizzare per visualizzare gli oggetti e lo sfondo dello schermo. L'animazione viene ottenuta memorizzando una porzione rettangolare dello schermo mediante il comando GET e copiandola in una nuova posizione tramite l'istruzione PUT. L'enunciato VIEW permette di limitare la visualizzazione grafica a una porzione rettangolare dello

schermo, mentre WINDOW consente di personalizzare il sistema di coordinate da utilizzare per specificare i punti. WINDOW consente di passare alternativamente tra due sistemi di coordinate.

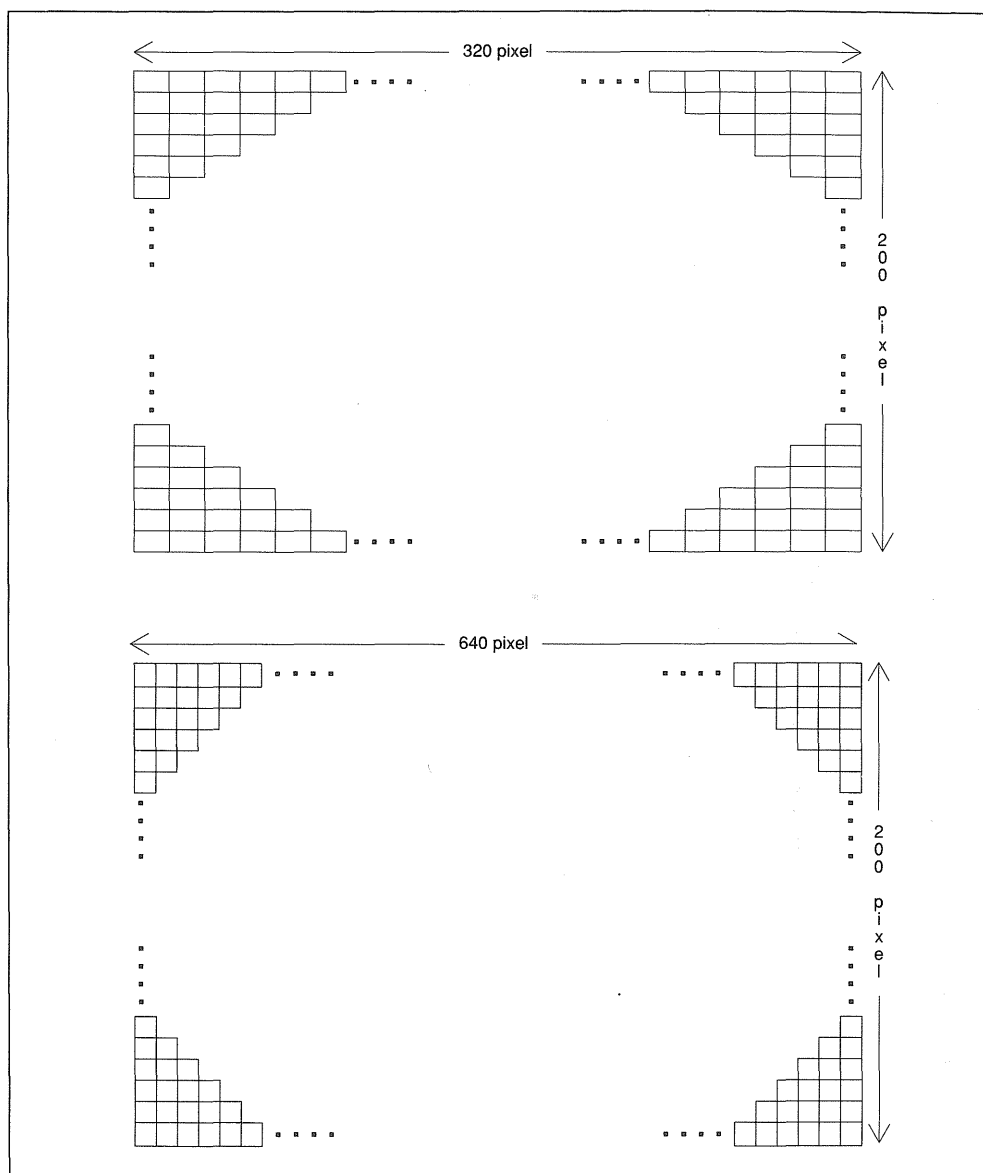



Figura 8.1 *Modalità grafica a media e alta risoluzione*

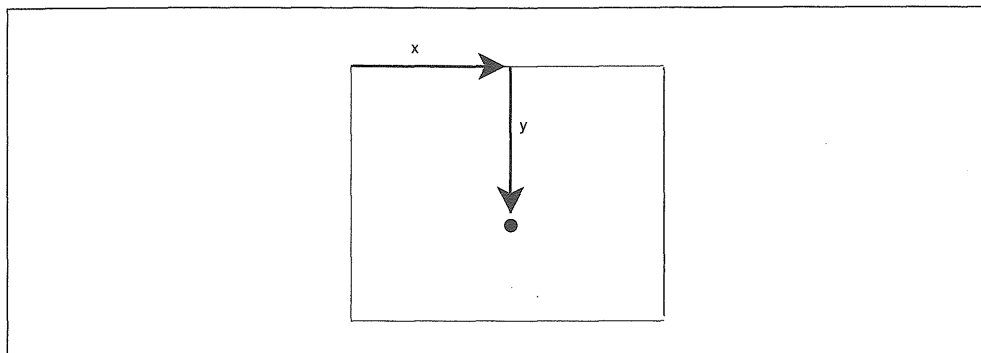



Figura 8.2 *Le coordinate di un punto*

Ogni modalità SCREEN viene identificata da un numero. La Tabella 8.1 riporta questi numeri. L'enunciato

```
SCREEN 0,1
```

specifica la modalità di testo, mentre l'enunciato

```
SCREEN n, 0
```

dove *n*>0, specifica la modalità grafica numero *n*. In modo testo, vengono visualizzati 40 caratteri per riga in modalità 1 e 80 caratteri per riga in modalità 2. Da questo momento in avanti, si presumerà che sia stata selezionata una modalità grafica. Il secondo parametro nell'enunciato SCREEN attiva o disattiva il colore sui monitor compositi.

Quando si usa una monitor monocromatico collegato a una scheda Hercules, si deve impartire il comando

```
SCREEN 3
```

per abilitare la grafica. Questa modalità è simile a quella attivata con SCREEN 2.

Tabella 8.1 *Modalità video*

Numero	Modalità video	Effetto
0	modo testo	solo testo
1	media risoluzione	testo e grafica
2	alta risoluzione	testo e grafica
3	grafica Hercules	testo e grafica

PUNTI, LINEE, RETTANGOLI E CERCHI

L'enunciato

```
PSET (x, y)
```

attiva il punto con coordinate (*x, y*), mentre l'enunciato

```
PRESET (x, y)
```

lo disattiva. Il comando

```
LINE (x1, y1)-(x2, y2)
```

traccia una linea tra il punto (*x1, y1*) e il punto (*x2, y2*). L'enunciato

```
LINE (x1, y1)-(x2, y2), , B
```

traccia un rettangolo i cui angoli opposti sono i punti (*x1, y1*) e (*x2, y2*). L'istruzione

```
LINE (x1, y1)-(x2, y2), , BF
```

traccia lo stesso rettangolo, ma questa volta pieno. L'enunciato

```
CIRCLE (x, y), r
```

traccia un cerchio con centro (*x, y*) e raggio *r*. La lunghezza del raggio corrisponde alla distanza orizzontale tra il centro e il limite del cerchio. Le Figure 8.3, 8.4, 8.5, 8.6 e 8.7 mostrano gli effetti generati da questi comandi in media risoluzione. Delle variazioni al comando CIRCLE consentono di tracciare ellissi e archi. Questa caratteristica è discussa nel Capitolo 10.

ULTIMO PUNTO INDIRIZZATO E COORDINATE RELATIVE

Dopo aver eseguito un qualunque comando grafico, esiste sempre un punto sullo schermo conosciuto come *ultimo punto indirizzato*. Questo è l'ultimo punto a cui un enunciato grafico ha fatto riferimento. Quando si esegue per la prima volta un programma, o si impartisce un comando SCREEN o CLS, l'ultimo punto indirizzato diventa il centro dello schermo.

Un'altra forma del comando LINE consente di omettere una delle coordinate dei punti. Quando si usa questa variante, QBasic utilizza, al posto delle coordinate

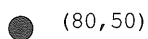



Figura 8.3 *Il risultato di PSET(80, 50)*

Figura 8.4 *Il risultato di LINE(20, 30)-(200, 70)*

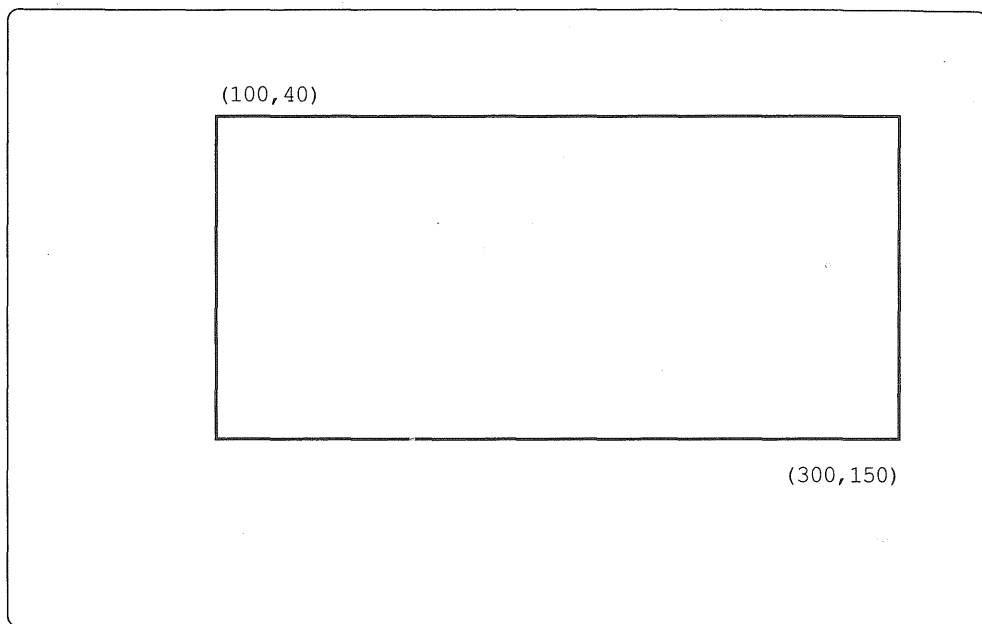



Figura 8.5 *Il risultato di LINE(300, 150)-(100, 40), ,B*

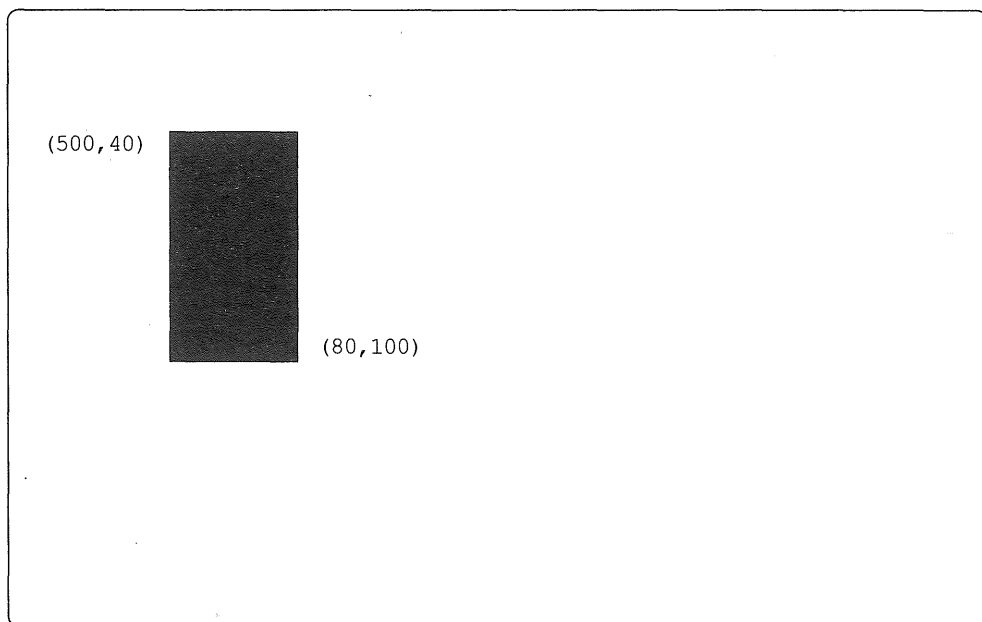



Figura 8.6 *Il risultato di LINE(50, 40)-(80, 100), , BF*

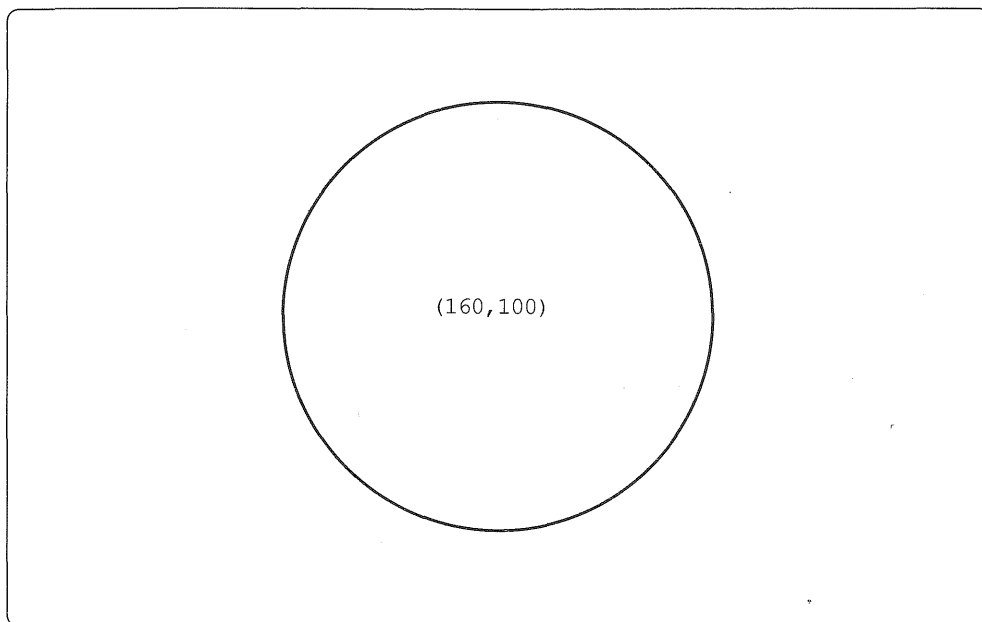



Figura 8.7 *Il risultato di CIRCLE(160, 100), 80*

omesse, l'ultimo punto indirizzato. Ad esempio, se l'ultimo punto indirizzato fosse il centro dello schermo, l'enunciato

```
LINE -(100, 50)
```

traccerebbe una linea dal centro dello schermo al punto (100,50).

Tabella 8.2 *Esempi dell'ultimo punto indirizzato e di coordinate relative*

Enunciato	Ultimo punto indirizzato
`CIRCLE (80,70), 30`	(80,70)
`LINE (0,0)-(40,50)`	(40,50)
`LINE (50,80)-STEP (20,10)`	(70,90)
`PSET (20,30)`	(20,30)
`PSET (10,20): CIRCLE STEP (30,40),10`	(40,60)

Nell'esempio precedente, per identificare i punti sono state usate delle coordinate assolute. Si possono tuttavia utilizzare delle coordinate *relative*; queste coordinate

vengono specificate in relazione all'ultimo punto indirizzato e sono precedute dalla parola chiave STEP. Se si utilizza il comando

```
STEP (r, s)
```

QBasic inizia dall'ultimo punto indirizzato, spostandosi di *r* punti in orizzontale (verso destra se *r* è positivo e verso sinistra se *r* è negativo) e di *s* punti in verticale (verso il basso se *s* è positivo, e verso l'alto se *s* è negativo). Quindi, se l'ultimo punto indirizzato avesse le coordinate assolute (a, b), il comando STEP(r,s) identificherebbe il punto con coordinate $(a+r, b+s)$. La Tabella 8.2 riporta alcuni enunciati e gli ultimi punti indirizzati dopo la loro esecuzione

IL COMANDO DRAW

L'enunciato DRAW mette a disposizione un piccolo linguaggio grafico che consente di disegnare delle figure tracciando una sequenza di linee, ognuna delle quali si estende dall'ultimo punto indirizzato a un punto specificato. Queste figure possono essere ruotate, colorate, duplicate e ridimensionate.

IL SOTTOCOMANDO M DELL'ENUNCIATO DRAW

L'enunciato

```
DRAW "M x,y"
```

traccia una linea retta dall'ultimo punto indirizzato al punto con coordinate (x,y). Dopo l'esecuzione del comando, il punto (x,y) diventa il nuovo ultimo punto indirizzato. Per esempio, gli enunciati

```
CLS: DRAW "M 200,50": DRAW "M 300,150"
```

generano una linea che si estende dal centro dello schermo fino al punto (200,50) e una linea compresa tra i punti (200,50) e (300,150). Gli ultimi due enunciati possono essere riuniti nell'unico comando

```
DRAW "M 200,50 M 300,150"
```

Se in un enunciato DRAW la lettera N precede la lettera M, l'ultimo punto indirizzato non cambia. Per esempio,

```
CLS: DRAW "NM 200,50 M 300,150"
```

traccia una linea che si estende dal centro dello schermo al punto (200,50) e un'altra dal centro al punto (300,150). Il punto (300,150) resta l'ultimo punto indirizzato. Se si antepone la lettera B alla lettera M in un enunciato DRAW, non viene tracciata nessuna linea, ma viene semplicemente cambiato l'ultimo punto indirizzato. Per esempio,

```
DRAW "BM 200,50 M 300,150"
```

traccia una singola linea dal punto (200,50) al punto (300,150). L'enunciato DRAW "BM *x,y*" viene usato per impostare il punto iniziale prima di tracciare una figura con il comando DRAW.

Le coordinate di un punto possono anche essere specificate in formato relativo. Se *r* e *s* sono due interi non negativi, l'enunciato

```
DRAW "M +r, s"
```

traccia una linea dall'ultimo punto indirizzato al punto che si trova *r* punti verso destra e *s* punti verso il basso. Ad esempio, in modalità grafica a media risoluzione,

```
CLS: DRAW "M+40,50"
```

traccia una linea dal punto (160,100) al punto (200,150). Gli enunciati DRAW "M-*r,s*", DRAW "M+*r,s*" e DRAW "M-*r,-s*" hanno interpretazioni analoghe. La presenza del segno + o - davanti alla prima coordinata indica che si stanno utilizzando delle coordinate relative.

Tutti gli enunciati seguenti riproducono la lettera X riportata in Figura 8.8:

```
CLS: DRAW "NM 210,50 NM 210,150 NM 110,150 M 110,50"
CLS: DRAW "BM 110,50 M 210,150 BM 210,150 M 110,150"
CLS: DRAW "BM 110,50 M +100,100 BM +0,-100 M -100,100"
```

I sottocomandi di direzione U, D, L, R, E, F, G e H dell'enunciato DRAW

Un enunciato nella forma

```
DRAW "U n"
```

dove *n* è un intero positivo, traccia una linea che parte dall'ultimo punto indirizzato e si estende di *n* punti verso l'alto. Questo comando è equivalente all'enunciato DRAW "M+0,-*n*". Gli altri sottocomandi di direzione sono illustrati nella Figura 8.9 e nella Tabella 8.3.

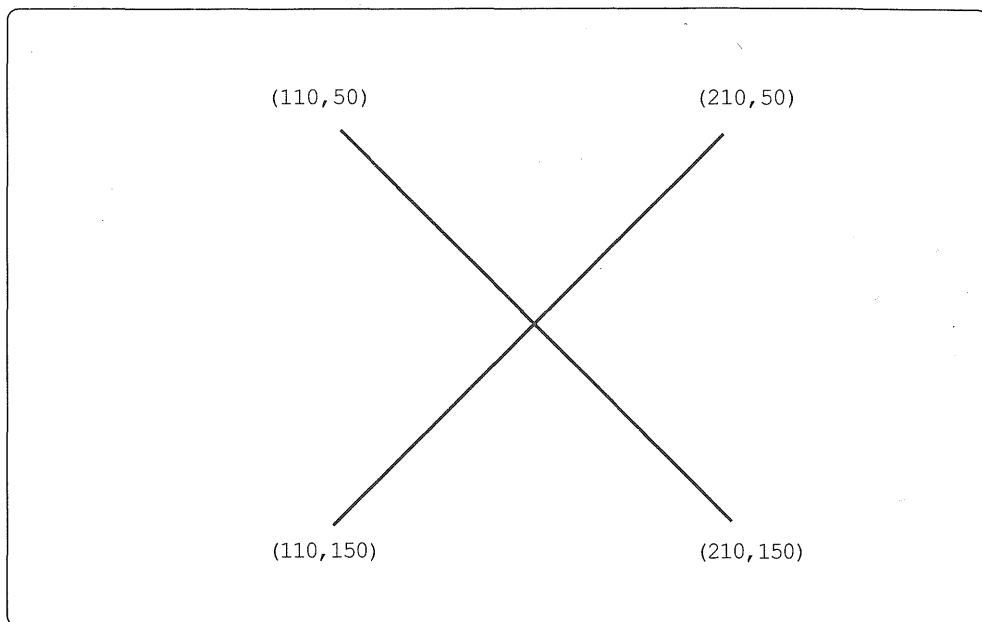



Figura 8.8 *La lettera X generata dal comando DRAW*

Se non viene specificato nessun valore dopo uno dei sottocomandi U, D, L, R, E, F, G o H, viene usato il valore di default 1. Il parametro *n* nei sottocomandi U, D, L, R, E, F, G e H può anche essere un intero negativo. Ad esempio, DRAW "U-9" equivale al comando DRAW "D9", e DRAW "E-9" ha lo stesso effetto di DRAW "G9".

Il Programma 8.1 consente di tracciare delle figure disegnando un punto alla volta in una qualsiasi direzione. Si possono premere i tasti U, D, L, R, E, F, G o H per spostarsi nella direzione corrispondente. Si prema il tasto Q per uscire dal programma. Se si preme un tasto diverso da quelli consentiti, il programma emette un segnale acustico. Il Programma 8.2 genera il cubo mostrato in Figura 8.10.

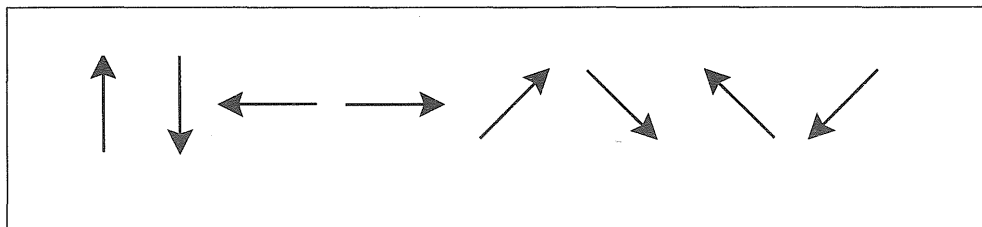



Figura 8.9 *Le otto direzioni usate dal comando DRAW*

Tabella 8.3 *I sottocomandi di direzione*

Sottocomando	Spostamento	Equivalente a
U *n*	*n* punti verso l'alto	M+0,-*n*
D *n*	*n* punti verso il basso	M+0,*n*
L *n*	*n* punti verso sinistra	M-*n*,0
R *n*	*n* punti verso destra	M+*n*,0
E *n*	*n* punti in alto a destra	M+*n*,-*n*
F *n*	*n* punti in basso a destra	M+*n*,*n*
G *n*	*n* punti in basso a sinistra	M-*n*,*n*
H *n*	*n* punti in alto a sinistra	M-*n*,-*n*

I prefissi N e B con i sottocomandi di direzione producono lo stesso effetto esaminato in precedenza. Per esempio, l'enunciato DRAW "BU20" sposta l'ultimo punto indirizzato di 20 unità verso l'alto senza tracciare una linea. La lettera X mostrata in Figura 8.8 può anche essere tracciata con uno degli enunciati seguenti:

```
CLS: DRAW "E50 G100 H50 F100"
CLS: DRAW "NE50 50 NG50 H50"
CLS: DRAW "BE50 G100 BU100 F100"
```

Programma 8.1 *Conversione dello schermo in area di disegno*

```
REM Conversione dello schermo in area di disegno [8-1]
SCREEN 1, 0     ' Usare SCREEN 3 con la Hercules
PRINT "Comandi di disegno: U D L R E F G H."
PRINT "Per uscire premere Q."
a$ = ""
DO WHILE a$ <> "Q"
  SELECT CASE a$
    CASE "U", "D", "L", "R", "E", "F", "G", "H", ""
      DRAW a$
    CASE ELSE
      PRINT CHR$(7);
  END SELECT
  a$ = UCASE$(INPUT$(1))
LOOP
END
```

Gli enunciati DRAW appena riportato sono stati scritti in lettere maiu
scole e includono degli spazi solamente per apparire più leggibili. Nessuna di queste convenzioni è necessaria. Ad esempio, l'enunciato che traccia il cubo nel Programma 8.2 potrebbe essere scritto nel modo seguente:

```
DRAW "l60u60r60d60e20u60g20e20l60g20"
```

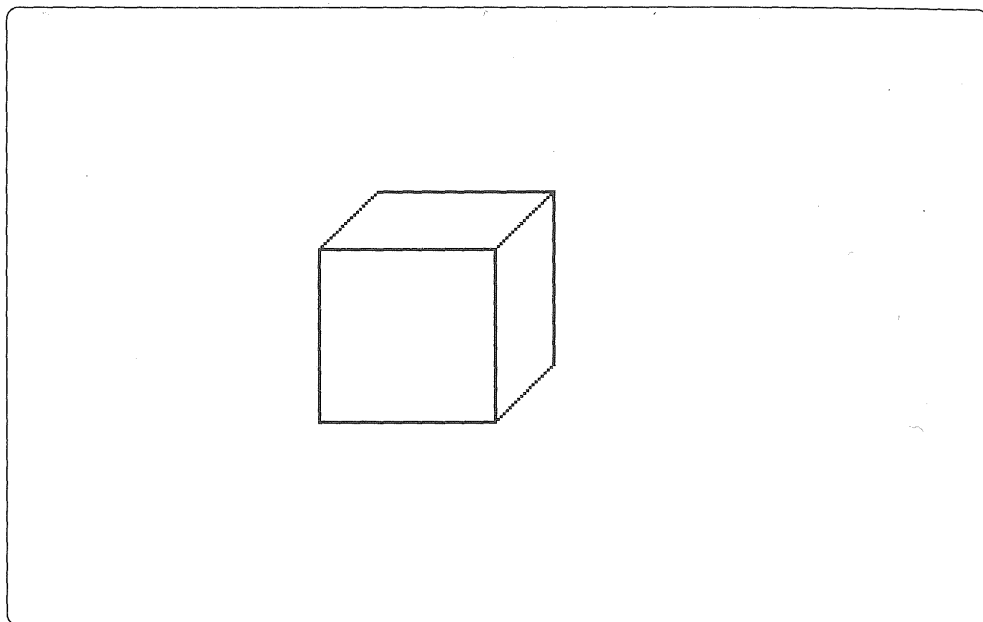



Figura 8.10 *Il cubo tracciato dal Programma 8.2*

Programma 8.2 *Disegno di un cubo*

```
REM Traccia un cubo  [8-2]
SCREEN 1, 0
DRAW "L60 U60 R60 D60 E20 U60 G20 E20 L60 G20"
END
```

IL SOTTOCOMANDO ANGLE

L'enunciato

```
DRAW "A n"
```

indica a QBasic di tracciare tutte le figure successive ruotate in senso antiorario di *n**90 gradi, dove *n* può essere 0, 1, 2 o 3. Il comando

```
DRAW "TA n"
```

indica a QBasic di tracciare tutte le figure successive ruotate di *n* gradi, dove *n* è compreso tra -360 e 360.

Il Programma 8.3 traccia il cubo riportato in Figura 8.3. Il cubo è stato ruotato in senso antiorario di 45 gradi.

IL SOTTOCOMANDO SCALE

Le figure create con gli enunciati DRAW possono essere ridimensionate usando il sottocomando Scale. L'enunciato

```
DRAW "S n"
```

indica a QBasic di tracciare tutti i segmenti successivi con una dimensione pari a *n*/4 di quella specificata, fino a quando non viene indicata un'altra scala. Il numero *n* può essere compreso tra 1 e 255. Ad esempio, l'enunciato DRAW "S80 U10 L15 D10 R15" equivale all'enunciato DRAW "U20 L30 D20 R30". Entrambi tracciano un rettangolo di altezza 20 e larghezza 30.

Programma 8.3 *Uso del sottocomando Angle*

```
REM Traccia un cubo ruotato  [8-3]
SCREEN 1, 0
DRAW "TA45 L60 U60 R60 D60 E20 U60 G20 E20 L60 G20"
END
```

USO DI VARIABILI E SOTTOSTRINGHE IN DRAW

In un enunciato DRAW è possibile utilizzare una variabile al posto della stringa. Ad esempio, l'enunciato DRAW "U20 L30 D20 R30" equivale a:

```
a$ = "U20 L30 D20 R30": DRAW a$
```

Si può anche utilizzare l'operatore + per combinare delle stringhe in un enunciato DRAW. Per esempio, il comando DRAW nel Programma 8.3 potrebbe essere sostituito con

```
cube$ = "L60 U60 R60 D60 E20 U60 G20 E20 L60 G20": DRAW "TA45 " + cube$
```

Finora, i numeri nelle stringhe del comando DRAW sono sempre stati delle costanti numeriche. È possibile inserire una variabile numerica in un enunciato DRAW convertendo il valore in stringa tramite la funzione STR$ e usando quindi l'operatore + per aggiungerlo alla stringa di DRAW. La Tabella 8.4 mostra alcuni esempi del comando DRAW e del formato equivalente con l'uso di variabili.

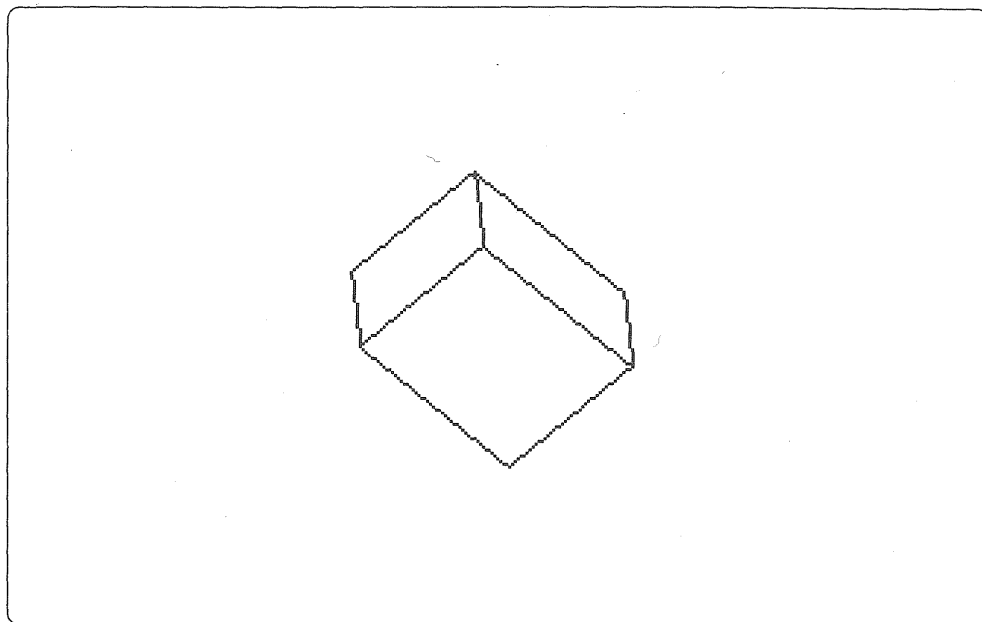



Figura 8.11 *Il cubo generato dal Programma 8.3*

Tabella 8.4 *Esempi del comando DRAW con l'uso di variabili*

Enunciato	Formato equivalente con le variabili
DRAW "U 20"	a=20: DRAW "U" + STR$(a)
DRAW "TA 45"	angolo=45: DRAW "TA" + STR$(angolo)
DRAW "NL5 D2"	a=5: b=2: DRAW "NL" + STR$(a) + "D" + STR$(b)
DRAW "M +40,50"	a=40: b=50: DRAW "M+" + STR$(a) + "," + STR$(b)

Il Programma 8.4 usa il sottocomando Scale per tracciare dei cubi di varie dimensioni. Il cubo non può essere contenuto interamente sullo schermo se la scala è maggiore di 180.

COLORE

La Tabella 8.5 mostra i 16 colori principali disponibili sui monitor a colori. Ciascun colore è identificato da un numero compreso tra 0 e 15. Questa sezione presume che si utilizzi la modalità grafica in media risoluzione. Più avanti in questo capitolo, verranno presentate le capacità più avanzate dei sistemi EGA e VGA.

Programma 8.4 *Uso delle variabili in una stringa DRAW*

```
REM Traccia dei cubi di diverse dimensioni  [8-4]
SCREEN 1, 0
INPUT "SCALA (1-255) "; scale
DRAW "S" + STR$(scale)
DRAW "BM +0,2 L3 U3 R3 D3 E1 U3 G1 E1 L3 G1"
END
```

Si può pensare a una palette come a una serie di quattro barattoli di vernice numerati da 0 a 3, ciascuno dei quali contenente una vernice diversa. La grafica in media risoluzione mette a disposizione due palette, numerate da 0 a 1, che possono essere utilizzate in qualsiasi momento. La Tabella 8.6 riporta i colori contenuti nei barattoli. Sullo schermo possono apparire solo 4 colori contemporaneamente, e tutti questi colori devo essere della stessa palette.

Tabella 8.5 *I 16 colori principali*

0	Nero	4	Rosso	8	Grigio	12	Rosso chiaro
1	Blu	5	Magenta	9	Blu chiaro	13	Magenta chiaro
2	Verde	6	Marrone	10	Verde chiaro	14	Giallo
3	Ciano	7	Bianco	11	Ciano chiaro	15	Bianco intenso

Il comando COLOR viene utilizzato per scegliere la palette da utilizzare come palette corrente, e per assegnare il colore di sfondo al barattolo numero 0. L'enunciato

```
COLOR b, p
```

dove *b* è un numero compreso tra 0 e 15 e *p* un numero tra 0 e 1, indica a QBasic di utilizzare la palette *p* e il colore *b* come colore di sfondo. Il testo appare sempre nel colore assegnato al barattolo 3 nella palette corrente. Tutte le figure generate da uno degli enunciati seguenti appariranno con il colore assegnato al barattolo *m* nella palette corrente:

```
PSET (x,Y), m
LINE (x1,y1)-(x2,y2), m
CIRCLE (x,y), r, m
DRAW "C m" + stringa
```

Se si omette il parametro *m* in uno di questi enunciati, QBasic traccia la figura usando il terzo colore della palette corrente.

Nota: In questo libro, si utilizza la dicitura colore *m* per indicare il colore assegnato al barattolo *m* nella palette corrente. Alcuni libri usano la parola attributo per far riferimento al colore *m*.

Tabella 8.6 *Le due palette standard*

Palette 0		Palette 1	
Barattolo n.	**Colore assegnato**	**Barattolo 2**	**Colore assegnato**
0	b (colore di sfondo)	0	b (colore di sfondo)
1	2 (verde)	1	3 (ciano)
2	4 (rosso)	2	5 (magenta)
3	6 (marrone)	3	7 (bianco)

Il Programma 8.5 disegna delle figure e scrive del testo in colori differenti. Il Programma 8.6 traccia il cubo mostrato precedentemente in Figura 8.10 usando il bianco come colore di sfondo e il magenta e il ciano per i lati.

Oltre a tracciare delle linee colorate, QBasic consente di riempire delle regioni con un colore selezionato dalla palette corrente. Una regione viene identificata specificando le coordinate di un punto all'interno della regione stessa e il colore del suo bordo. Se il punto (x,y) si trovasse all'interno di una regione con bordo di colore *b*, l'enunciato

```
PAINT (x,y), m, b
```

riempirebbe la regione con il colore *m* della palette corrente. Per esempio, il Programma 8.7 crea un bersaglio. Il cerchio più interno è nero e gli altri tre sono di colore ciano, magenta e bianco.

L'enunciato DRAW dispone di un sottocomando di riempimento che può essere incluso nella stringa. Il comando

```
DRAW "P m, b"
```

ha lo stesso effetto dell'enunciato PAINT (x,y), *m*, *b* dove il punto (x,y) è l'ultimo punto indirizzato.

Il Programma 8.8 crea un cubo con la parte frontale rossa, quella superiore marrone e quella di destra verde, usando uno sfondo bianco. Le ultime tre righe del programma utilizzano la lettera B nel comando DRAW per spostare l'ultimo punto indirizzato all'interno di una delle facce del cubo prima di eseguire l'operazione di riempimento.

Programma 8.5 *Visualizzazione di testo e figure a colori*

```
REM Uso dei colori  [8-5]
SCREEN 1, 0
COLOR 8, 0                    'Sfondo grigio, palette 0
PRINT "Ciao"                  'Testo marrone
PSET (50, 50)                 'Punto marrone
PSET (100, 100), 1            'Punto verde
LINE (10, 20)-(30, 40), 2     'Linea rossa
CIRCLE (100, 100), 50, 3      'Cerchio marrone
PSET (150, 150)
DRAW "C1 R6 U5 L6 D5"         'Quadrato verde
END
```

Programma 8.6 *Disegno di un cubo a colori*

```
REM Traccia il cubo della Figura 8-10 con colori diversi  [8-6]
SCREEN 1, 0
COLOR 7, 1
CLS
DRAW "C1 L60 U60 R60 D60 C2 E20 U60 G20 E20 L60 G20"
END
```

STILE E TASSELLATURA

QBasic consente di definire altre due caratteristiche per sostituire o completare i colori. Si può cambiare lo stile delle linee per tracciare, ad esempio, delle linee punteggiate o tratteggiate, ed è possibile definire dei modelli tassellati da utilizzare come motivo di riempimento delle figure.

Programma 8.7 *Uso del comando PAINT*

```
REM Crea un bersaglio  [8-7]
SCREEN 1, 0
COLOR 0, 1
FOR n = 1 TO 4
  CIRCLE (160, 100), 20 * n, 3
NEXT n
FOR n = 1 TO 3
  PAINT (170 + 20 * n, 100), n, 3
NEXT n
END
```

Per cambiare lo stile delle linee è necessario saper convertire un numero binario (composto da una sequenza di valori 0 e 1) in formato esadecimale (un numero che inizia con &H ed è composto da cifre comprese tra 0 e 9 e da lettere comprese tra A

e F). Ogni volta che si deve specificare lo stile di una linea, il numero binario viene fornito come stringa composta da 16 valori uguali a zero o uno. Per convertire una stringa binaria in formato esadecimale, si proceda nel modo seguente:

1. suddividere la stringa di 16 cifre in quattro gruppo di quattro cifre ciascuno;
2. sostituire ogni gruppo di quattro cifra con il numero o la lettera riportata nella Tabella 8.7;
3. aggiungere &H all'inizio della stringa di quattro caratteri risultante.

Programma 8.8 *Il sottocomando Paint dell'enunciato DRAW*

```
REM Colora un cubo  [8-8]
SCREEN 1, 0
COLOR 7, 0
a$ = "L60 U60 R60 D60 E20 U60 G20 E20 L60 G20"
DRAW a$
DRAW "BF5 P 2,3"
DRAW "BR60 P 1,3"
DRAW "BH10 P 3,3"
END
```

Per esempio, la stringa 0101111110100010 viene suddivisa in 0101 1111 1010 0010, e risulta in &H5FA2.

La Figura 8.12 mostra quattro stili di linea. Lo stile di ciascuna linea viene identificato da una stringa di 16 cifre che viene ripetuta quante volte risulta necessario. Per tracciare una linea con un determinato stile dal punto (*a,b*) al punto (*c,d*), si considerino i pixel situati sulla linea retta che si estende tra i due punti specificati. Si supponga di iniziare con il punto (*a,b*) e di attivare, dei primi 16 pixel, il primo, il quinto, il decimo e il quattordicesimo. Questa combinazione può essere rappresentata con la stringa

```
1000100001000100
```

Tabella 8.7 *Numeri binari e loro equivalenti esadecimali*

0000	0	0100	4	1000	8	1100	C
0001	1	0101	5	1001	9	1101	D
0010	2	0110	6	1010	A	1110	E
0011	3	0111	7	1011	B	1111	F

Spostandosi da sinistra verso destra, questa stringa contiene il valore 1 nelle posizioni 1, 5, 10 e 14, e degli zeri in tutte le altre. Questa stringa è la rappresentazione binaria

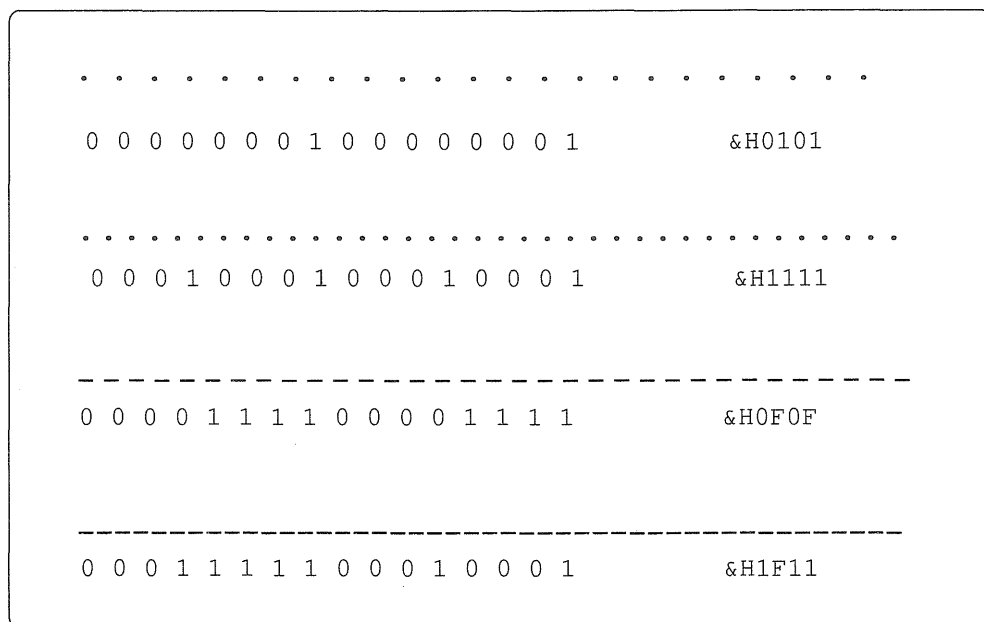



Figura 8.12 *Stili di linea*

del numero esadecimale &H8844. L'enunciato

```
LINE (a,b)-(c,d), , , &H8844
```

traccia una linea dal punto (*a*, *b*) al punto (*c*, *d*), partendo da (*a*, *b*) con lo stile di linea specificato e ripetendo lo stesso stile per ogni punto fino a raggiungere (*c*, *d*). In generale, se *s* è la rappresentazione esadecimale di una stringa binaria di 16 cifre, l'enunciato

```
LINE (a,b)-(c,d), , , s
```

traccia la linea da (*a*, *b*) a (*c*, *d*) utilizzando lo stile specificato dal valore esadecimale *s*. Il numero *s* identifica lo stile della linea. La Figura 812 mostra alcuni stili di linea con la stringa binaria associata. Si possono inserire tra le virgole i parametri *m* e/o B per ottenere i colori o un rettangolo.

Il Programma 8.9 traccia una linea usando l'ultimo stile di quelli rappresentati in Figura 8.12. La funzione POINT restituisce il valore 3 se il pixel indicato è attivo, e il valore 0 in caso contrario. La combinazione di zero e tre è uguale a quella di zero e uno che appare nella rappresentazione binaria di &H1F11.

Programma 8.9 *Impostazione degli stili delle linee*

```
REM Determina il motivo di una linea con stile  [8-9]
SCREEN 1, 0
LINE (20, 170)-(300, 170), , , &H1F11
FOR n = 0 TO 15
  PRINT POINT(20 + n, 170);
NEXT n
END

[Esecuzione]
 0  0  0  3  3  3  3  3  0  0  0  3  0  0  0  3
```

La Figura 8.13 mostra due esempi di regioni delimitate riempite con un motivo tassellato. Il metodo per la definizione dei motivi di riempimento in media risoluzione è diverso da quello per l'alta risoluzione. Questa sezione discute entrambi i casi separatamente.

In modalità grafica ad alta risoluzione, ciascun motivo di riempimento viene definito da 8 pixel in orizzontale e da 1 a 64 pixel in verticale. Si può associare ogni riga del motivo a una stringa composta da 8 cifre uguali a zero o uno, dove uno indica un pixel attivo e zero un pixel non attivo. Ogni stringa binaria di 8 cifre corrisponde a un intero compreso tra 0 e 255.

La Figura 8.14 mostra il motivo di riempimento usato nella Figura 8.13a, insieme alle stringhe binarie e ai valori decimali corrispondenti. La tassellatura nella Figura 8.14 viene specificata dalla stringa

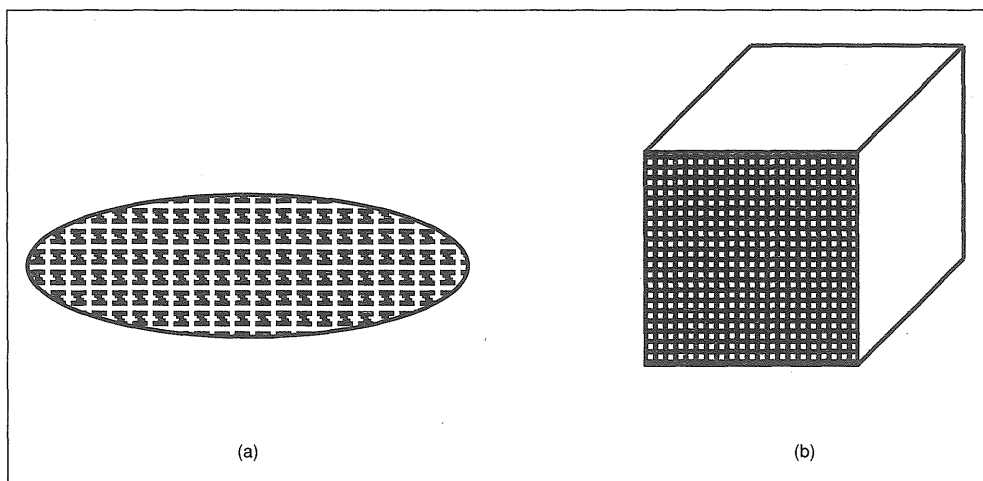



Figura 8.13 *Esempi di motivi di riempimento*

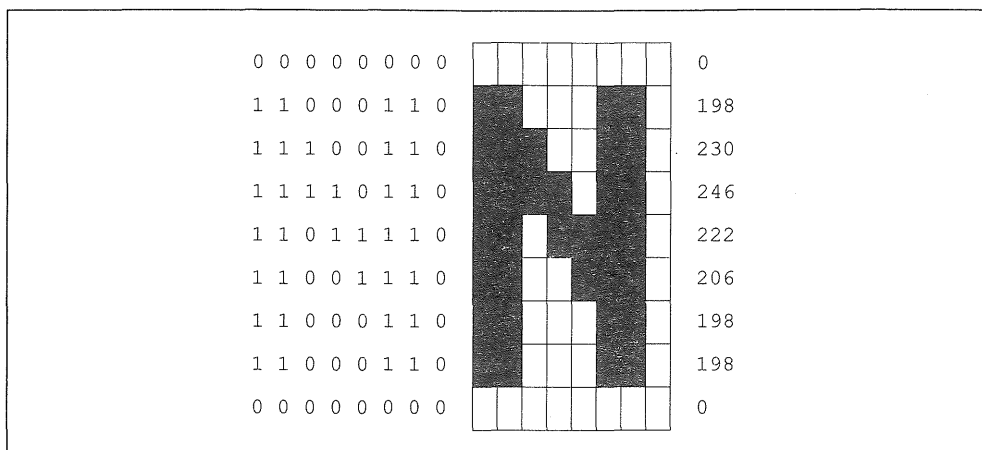



Figura 8.14 *La tassellatura usata nella Figura 8.13a*

```
t$=CHR$(0)+CHR$(198)+CHR$(230)+CHR$(246)+CHR$(222)+CHR$(206)+CHR$(198)+CHR$(198)+CHR$(0)
```

Il Programma 8.10 utilizza il motivo di riempimento della Figura 8.14 per riempire un cerchio con la stessa tassellatura dell'ellisse riportata in Figura 8.13a.

In modalità grafica a media risoluzione, ciascun motivo di riempimento viene definito da 4 pixel in orizzontale e da 1 a 64 pixel in verticale. Un motivo viene definito specificando il numero del barattolo (0, 1, 2 o 3) da usare per colorare ciascun pixel. Bisogna quindi convertire il numero scelto in notazione binaria: 0 diventa 00, 1 diventa 01, 2 diventa 10 e 3 diventa 11. A questo punto, si può associare ogni riga del motivo a una stringa composta da 8 cifre uguali a zero o uno, dove uno indica un pixel attivo e zero un pixel non attivo. Ogni stringa binaria di 8 cifre corrisponde a un intero compreso tra 0 e 255.

Programma 8.10 *Disegno di un cerchio pieno*

```
REM Genera un cerchio riempito con N  [8-10]
SCREEN 2, 0
CIRCLE (100, 75), 90
t$ = CHR$(0) + CHR$(198) + CHR$(230) + CHR$(246) + CHR$(222)_
    + CHR$(206) + CHR$(198) + CHR$(198) + CHR$(0)
PAINT (100, 75), t$
END
```

La Figura 8.15 mostra il motivo di riempimento usato nella Figura 8.13b insieme alle stringhe binarie e ai valori decimali corrispondenti. La tassellatura nella Figura 8.15 viene specificata dalla stringa

```
t$=CHR$(70)+CHR$(85)+CHR$(78)
```

In entrambe le modalità grafiche (ad alta e bassa risoluzione), se il motivo di riempimento ha *r* righe e gli interi associati sono *n1, n2, ..., nr*, la stringa

```
t$=CHR$(n1)+CHR$(n2)+...+CHR$(nr)
```

specifica la tassellatura. Se una regione definita contiene il punto (*x,y*) al suo interno e il suo bordo ha colore *b*, l'enunciato

```
PAINT (x,y), t$, b
```

riempie la regione con il motivo di riempimento definito da *t$*.

Programma 8.11 *Disegno di un cubo pieno*

```
REM Un cubo con una faccia piena  [8-11]
SCREEN 1, 0
COLOR 0, 0
DRAW "L60 U60 R60 D60 E20 U60 G20 E20 L60 G20"
t$ = CHR$(70) + CHR$(85) + CHR$(78)
PAINT (150, 90), t$
END
```

Il Programma 8.11 usa la tassellatura riportata in Figura 8.15 per ottenere il motivo di riempimento mostrato in Figura 8.13b.

In modalità grafica a media risoluzione, l'enunciato PAINT (*x,y*), *t$, b* non sempre riesce a riempire una regione già riempita con un altro motivo o con un colore. Una soluzione generale a questo problema è possibile, se il bordo della regione in questione non ha un colore di sfondo (cioè *b* non è uguale a 0). In questo caso, l'enunciato PAINT (*x,y*), 0, *b* dovrebbe precedere il comando PAINT mostrato in precedenza. Il primo enunciato PAINT cancella la regione lasciando solo il suo contorno e consentendo quindi al comando PAINT successivo di eseguire il correttamente l'operazione di riempimento.

0	0	0	0	0	0	0	0	1	0	1	2	70
1	1	0	0	0	1	1	0	1	1	1	1	85
1	1	1	0	0	1	1	0	1	0	3	2	78

Figura 8.15 *La tassellatura usata nella Figura 8.13b*

Il Programma 8.12 mostra un esempio di questo problema. Per risolverlo, si dovrebbe inserire l'enunciato PAINT (30,50),0,3 prima di PAINT (30,50),*t$*,3.

Il Programma 8.13 riempie l'intero schermo con un motivo di riempimento fornito dall'utente tramite una stringa.

Programma 8.12 *Riempimento di una figura già piena*

```
REM Disegna un rettangolo rosso e lo colora con delle strisce   [8-12]
SCREEN 1, 0
COLOR 0, 0
LINE (10, 10)-(50, 90), , B
PAINT (30, 50), 2, 3
t$ = CHR$(255) + CHR$(255) + CHR$(0) + CHR$(0) + CHR$(170) + CHR$(170)_
     + CHR$(0) + CHR$(0)
PAINT (30, 50), t$, 3
END
```

ANIMAZIONE

L'animazione delle immagini viene ottenuta memorizzando una porzione rettangolare dello schermo mediante il comando GET e copiandola in una nuova posizione tramite l'istruzione PUT.

L'istruzione grafica GET inserisce una copia di una porzione rettangolare dello schermo in memoria. Questa porzione rettangolare viene identificata specificando le coordinate degli angoli in alto a sinistra (*x&,y1*) e in basso a destra (*x2,y2*), come mostrato in Figura 8.16. Le informazioni salvate consistono del numero del barattolo associato a ciascun punto della regione. Ci si ricordi che il colore di un punto viene determinato dalla palette e dal numero di barattolo selezionato. L'enunciato

```
GET (x1,y1)-(x2,y2), nomeArray
```

memorizza una descrizione del rettangolo specificato in *nomeArray*. Si può usare un array numerico di qualsiasi tipo. Per semplicità, la parte rimanente di questa sezione prende in considerazione solamente gli array numerici interi.

Bisogna innanzi tutto dimensionare l'array con l'enunciato

```
DIM nomeArray(n)
```

(Normalmente, gli array con una dimensione *n* minore di 11 non deve essere necessariamente dimensionato. Tuttavia, ciò non vale se l'array viene utilizzato per la prima volta con un enunciato GET). In modalità grafica a media risoluzione, per determinare la dimensione *n* dell'array intero si proceda nel modo seguente:

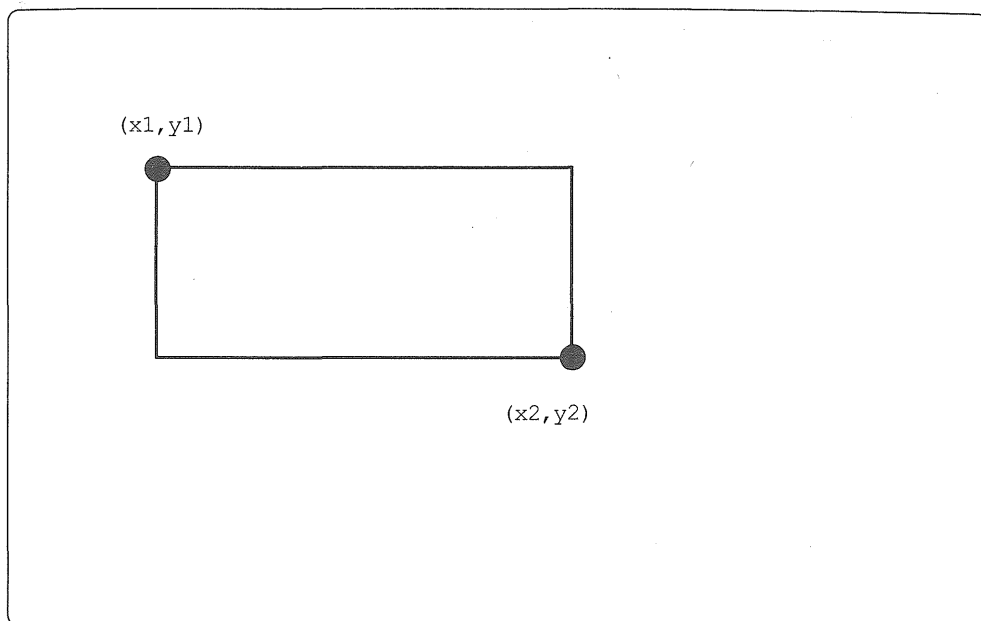



Figura 8.16 *Una regione rettangolare catturata da un enunciato GET*

Programma 8.13 *Diversi tipi di motivi di riempimento*

```
REM Esperimenti con i motivi di riempimento  [8-13]
SCREEN 1, 0
COLOR 0, 0
INPUT "Digitare una stringa di caratteri: ", t$
PAINT (5, 5), t$
END

[Esecuzione]
Digitare una stringa di caratteri: FUN
```

{Il motivo generato è uguale a quello che appare nel cubo rappresentato in Figura 8-13b}

1. assegnare ad h il numero di pixel del lato orizzontale del rettangolo, e a v il numero di pixel del lato verticale. Quindi,

   $h = x2 - x1 + 1$

   e

   $v = y2 - y1 + 1$

2. calcolare il numero (2*b+7)/8, moltiplicare la parte intera per v e sommare 4. In questo modo si ottiene il risultato b;
3. il valore di n deve essere almeno (b/2)-1.

In modalità grafica ad alta risoluzione, la procedura per determinare il valore di n è la stessa, ad eccezione del fatto che si deve sostituire il numero 2 del punto 2 con il numero 1.

Il numero n nel Programma 8.14 viene determinato nel modo seguente:

1. con gli stessi presupposti del punto 1 della procedura precedente, si ha:

   b=25-4+1=22

   v=60-20+1=41
2. 2*b+7=2*22+7=51

   51/8=6,375, la cui parte intera è 6

   b=6*v+4=6*41=250
3. n=(250/2)-1=124

L'istruzione grafica PUT viene normalmente usata in combinazione con il comando GET. Si supponga che un enunciato GET abbia memorizzato una porzione rettangolare dello schermo in un array. L'enunciato

```
PUT (x,y), nomeArray, PSET
```

visualizza una copia esatta della porzione rettangolare contenuta in *nomeArray*, posizionando l'angolo superiore di sinistra nel punto (x,y). Il Programma 8.15 visualizza H_2O sullo schermo. Il numero 2 appare come indice della lettera H.

Il Programma 8.16 disegna un camion, come mostrato in Figura 8.17, e lo sposta lungo lo schermo. La regione rettangolare scelta è leggermente più grande di quella necessaria per includere il camion. Nella parte a sinistra, la regione contiene una piccola porzione vuota che serve per cancellare, ogni volta che viene visualizzata sullo schermo una nuova immagine del camion, la parte che rimane del disegno precedente.

La parola PSET, che appare alla fine degli enunciati PUT, viene considerata l'azione dell'enunciato. Ci sono altri quattro tipi di azioni possibili: PRESET, AND, OR e XOR. Si considereranno ora gli effetti causati da queste azioni in modalità ad alta risoluzione.

Programma 8.14 *Dimensionamento di un array da usare con GET*

```
REM Cattura di una porzione rettangolare dello schermo  [8-14]
SCREEN 1, 0
n = 124
DIM a%(n)
GET (4, 20)-(25, 60), a%
END
```

Programma 8.15 *Uso dell'istruzione PUT*

```
REM Visualizza la formula chimica dell'acqua  [8-15]
SCREEN 1, 0
LOCATE 1, 1
PRINT "2"
DIM two%(9)
GET (0, 0)-(7, 7), two%
CLS
LOCATE 1, 1
PRINT "H O"
PUT (8, 4), two%, PSET
END
```

Programma 8.16 *Una dimostrazione di animazione*

```
REM Sposta un camion attraverso lo schermo  [8-16]
SCREEN 1, 0
CALL DrawTruck
DIM truck%(629)
GET (20, 21)-(119, 70), truck%
CLS
FOR n = 1 TO 200
  PUT (n, 100), truck%, PSET
NEXT n
END

SUB DrawTruck
  CIRCLE (105, 60), 10          'Disegna la ruota davanti
  PAINT (105, 60), 3            'Colora la ruota davanti
  CIRCLE (35, 60), 10           'Disegna la ruota dietro
  PAINT (35, 60), 3             'Colora la ruota dietro
  LINE (21, 21)-(101, 40), , BF 'Disegno il retro del camion
  LINE (21, 40)-(119, 60), , BF 'Disegna il cofano
END SUB
```

In modalità grafica ad alta risoluzione, ogni punto può essere bianco o nero. Si supponga che un enunciato GET abbia registrato una porzione rettangolare dello schermo in un array. Ognuna delle azioni ha effetto sulla regione rettangolare il cui

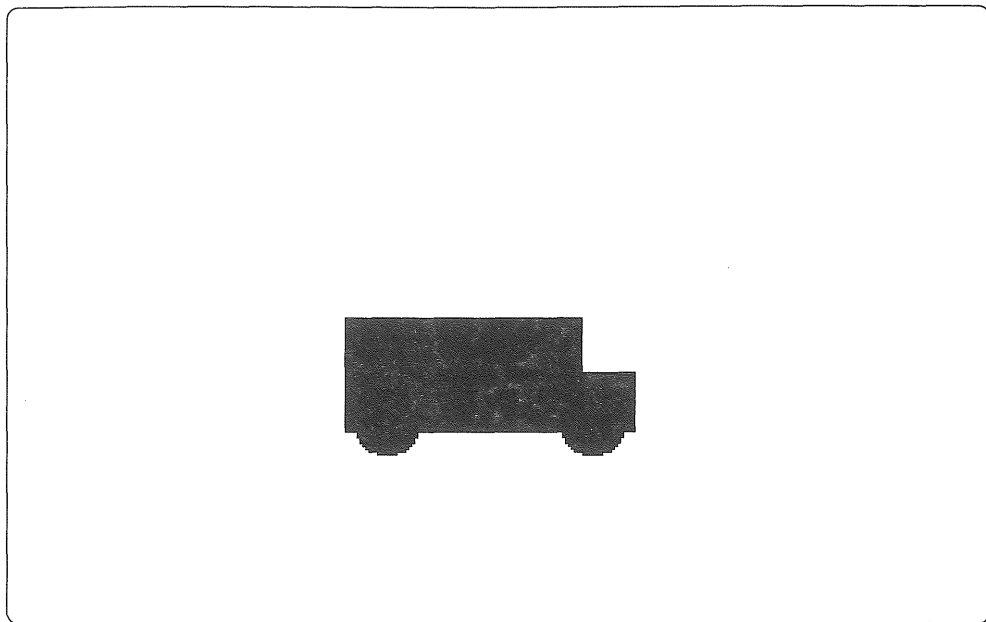



Figura 8.17 *Il camion creato dal Programma 8.16*

angolo in alto a sinistra corrisponde al punto (*x,y*) e la cui dimensione è uguale a quella del rettangolo originale.

L'enunciato

```
PUT (x,y), nomeArray, PRESET
```

visualizza un'immagine inversa di quella originale. In altre parole, ogni punto originariamente bianco diventa nero e viceversa.

Le altre tre azioni agiscono su qualunque immagine si trova già nella parte rettangolare dello schermo sui cui agirà l'istruzione PUT. L'azione AND genera un punto bianco solamente se in quella posizione esiste già un punto bianco e se anche il punto corrispondente dell'immagine in fase di trasferimento è bianco. L'azione OR genera un punto bianco se il punto già esistente e/o quello dell'immagine in corso di trasferimento è bianco. L'azione XOR, infine, genera un punto bianco se il punto già esistente o quello dell'immagine da trasferire è bianco, ma non entrambi. Se non viene specificata nessuna azione nel comando PUT, viene utilizzato XOR.

XOR viene comunemente usato nelle animazioni. Se si inserisce un'immagine sopra se stessa usando l'azione XOR, l'immagine viene cancellata e viene ripristinato lo

sfondo originale. Il Programma 8.17 mostra una palla che si sposta sullo schermo. Si prema un tasto qualsiasi per interrompere il programma.

Programma 8.17 *Uso dell'azione XOR*

```
REM Sposta una palla sullo schermo  [8-17]
DIM ball%(10)
t0 = TIMER
FOR i = 1 TO 500: NEXT i
loopsPerSec = 500 / (TIMER - t0)
'loopsPerSec= consente di ottenere una pausa di 0,004 secondi
SCREEN 2, 0
'Crea la palla
CIRCLE (20, 20), 4
PAINT (20, 20), 3
GET (16, 16)-(24, 24), ball%
'Specifica la posizione iniziale della palla
'e la incrementa per trovare la nuova posizione
horizontal = 16
vertical = 16
hIncrement = 1
vIncrement = 1
PRINT " Premere un tasto qualsiasi per uscire."
DO WHILE INKEY$ = ""
  'Se la palla si trova vicino a un bordo dello schermo
  'inverte il valore dell'incremento
  IF vertical > 190 OR vertical < 1 THEN vIncrement = -vIncrement
  IF horizontal > 630 OR horizontal < 1 THEN hIncrement = -hIncrement
  'Salva la posizione precedente per poter cancellare
  'successivamente la balla mediante il comando PUT
  oldHorizontal = horizontal
  oldVertical = vertical
  'Calcola la nuova posizione
  vertical = vertical + vIncrement
  'Cancella la palla dalla posizione precedente
  PUT (oldHorizontal, oldVertical), ball%, XOR
  'Inserisce la palla nella nuova posizione
  PUT (horizontal, vertical), ball%, XOR
  'Controlla la velocità della palla
  FOR i = 1 TO loopsPerSec * .004: NEXT i  'Ritardo di 0,004 secondi
LOOP
END
```

Se al Programma 8.17 si aggiunge, prima del ciclo DO WHILE, l'istruzione seguente

```
LINE (320,0)-(410,199), , BF
```

verrà tracciato un rettangolo verticale pieno nella parte centrale dello schermo. Quando la palla passa attraverso il rettangolo, quest'ultimo resta inalterato.

In modalità grafica a media risoluzione, l'azione PSET genera una copia esatta del rettangolo originale da visualizzare (a meno che non sia stata cambiata la palette corrente; in questo caso, i punti con colore *m* della palette precedente acquistano il colore *m* della nuova palette). L'azione PRESET visualizza un'immagine inversa; i punti con colore 3 acquistano il colore 0 della palette corrente, e viceversa. Analogamente, ai punti di colore 2 viene assegnato il colore 3 e viceversa.

Ad ogni punto dello schermo viene associato un numero di barattolo. Questo numero è uguale a 0 fino a quando il punto non viene elaborato da un comando grafico o da un enunciato PRINT. Le altre tre azioni di PUT hanno effetto sui colori che si trovano già nella porzione rettangolare dello schermo su cui agisce PUT. Il nuovo numero di barattolo per ciascun punto viene determinato sulla base del numero del barattolo precedente, del numero di barattolo del punto corrispondente nel rettangolo memorizzato e dall'azione. I numeri di barattolo risultanti (che determinano i colori) sono riportati nella Tabella 8.8.

Si possono anche ottenere i numeri di barattolo risultanti se si è in grado di utilizzare gli operatori logici con i numeri in rappresentazione binaria. Per esempio, per determinare il risultato della combinazione dei barattoli 2 e 3 con l'azione XOR, si scrivano i due numeri in notazione binaria e si applichi l'operatore logico XOR. I numeri 2 e 3 diventano, rispettivamente, 10 e 11. Dato che 10 XOR 11 restituisce il risultato 01, o 1, l'azione XOR sui barattoli 2 e 3 fornisce il barattolo 1.

Si può utilizzare la funzione POINT in un'animazione per determinare se due oggetti stanno per scontrarsi. Il valore di

```
POINT (x,y)
```

corrisponde al numero di barattolo del punto con coordinate (*x,y*). In modalità ad alta risoluzione, ai punti neri è associato il barattolo 0 e ai punti bianchi il barattolo 1.

Tabella 8.8 *Numeri di barattolo risultanti dalle azioni AND, OR e XOR*

		Numero di barattolo memorizzato		
Numero di barattolo precedente		**0123**	**0123**	**0123**
	0	0000	0123	0123
	1	0101	1133	1032
	2	0022	2323	2301
	3	0123	3333	3210
		AND	**OR**	**XOR**

Il Programma 8.18 crea una barriera di lunghezza e posizione definita dall'utente e genera una palla che si sposta lungo lo schermo. La Figura 8.18 mostra la barriera e la palla. Se la palla colpisce la barriera, la sua direzione viene invertita. Il programma ripete il ciclo DO fino a quando la palla raggiunge il bordo di destra o di sinistra dello schermo. La funzione POINT rileva gli oggetti che si trovano sulla strada della palla e informa il programma se ne deve cambiare la direzione. L'angolo in alto a sinistra del rettangolo che contiene la palla si estende per tre pixel a sinistra e tre pixel sopra il centro della palla. Se la lunghezza della barriera è minore di 100, la palla può attraversare l'intero schermo; in caso contrario, la palla si sconterà sempre con la barriera.

USO DI UNA SCHEDA EGA

A una scheda EGA si possono collegare tre tipi di monitor: EGM (Enhanced Graphics Monitor), RGB (un monitor a colori analogico le cui lettere indicano i tre colori fondamentali Red, Green e Blu) o uno monocromatico.

Si possono sfruttare interamente le capacità di una scheda EGA se si utilizza un monitor EGM; in questo caso, sono disponibili 64 colori e altre tre modi di risoluzione. Se si dispone di un monitor RGB, si possono cambiare i colori assegnati ai barattoli nelle palette delle due modalità grafiche a media risoluzione scegliendo quattro dei 16 colori disponibili; in modalità ad alta risoluzione, si possono specificare i colori di primo piano e di sfondo. Con un monitor monocromatico, infine, si può utilizzare la grafica con una modalità SCREEN speciale. Segue una descrizione di tutte le modalità SCREEN disponibili in QBasic:

- Modalità SCREEN 1 (risoluzione grafica 320x200, 40 caratteri per riga)

  monitor RGB: la situazione è identica a quella esaminata per la scheda CGA, ad eccezione del fatto che si può usare l'enunciato PALETTE per cambiare i colori assegnati alle palette. Se *m* è un numero compreso tra 0 e 3 e *c* un numero compreso tra 0 e 15, il comando PALETTE *m,c* cambia il colore nel barattolo *m* della palette corrente con il colore *c*, come mostrato nella Tabella 8.5. Si noti che se si seleziona una palette con il comando COLOR, vengono ripristinati i colori di default assegnati a quella palette. Il Programma 8.19 imposta la modalità SCREEN 1 con colore di sfondo blu chiaro e colori di primo piano giallo, magenta e verde chiaro;

  monitor EGM: la situazione è identica a quella vista per i monitor RGB, ad eccezione del fatto che i comandi COLOR e PALETTE mettono a disposizione 64 colori possibili per il colore di sfondo *b* e il colore *c*.

Programma 8.18 *Uso della funzione POINT per rilevare una collisione*

```
REM Sposta una palla sullo schermo con una barriera  [8-18]
DEFINT D, X
SCREEN 1, 0
t0 = TIMER
FOR i = 1 TO 500: NEXT i
loopsPerSec = 500 / (TIMER - t0)
'Crea la palla
DIM ball%(9)
CIRCLE (3, 100), 2
PAINT (3, 100), 3
GET (0, 97)-(6, 103), ball%
'Richiede la posizione e la dimensione della barriera
INPUT "Posizione ( da 10 a 300): ", position
INPUT "Dimensione (da 1 a 199): ", length
CLS
LINE (position, 0)-(position + 20, length), , BF
'Sposta la palla sullo schermo. Se la palla colpisce la
'barriera, inverte la direzione
d = 1           '1=destra, -1=sinistra
x = 3           'x-coordinata del centro della palla
DO
  IF POINT(x + 4, 99) <> 0 THEN d = -1
  x = x + d
  PUT (x - 3, 97), ball%, PSET
  FOR i = 1 TO loopsPerSec * .004: NEXT i
LOOP UNTIL x > 315 OR (x < 4 AND d = -1)
END
```

- Modalità SCREEN 2 (risoluzione grafica 640x200, 80 caratteri per riga)

  monitor RGB: la situazione è identica a quella esaminata per la scheda CGA, ad eccezione del fatto che si può usare l'enunciato PALETTE per impostare sia il colore di primo piano che quello di sfondo su uno dei 16 colori standard disponibili. Se *c* è un numero compreso tra 0 e 15, il comando PALETTE 0,*c* seleziona il colore *c* come colore di sfondo e PALETTE 1,*c* seleziona il colore *c* come colore di primo piano. Il Programma 8.20 imposta la modalità SCREEN 2 con colore di sfondo blu chiaro e colore di primo nero;

  monitor EGM: si possono usare due colori qualsiasi dei 64 disponibili come sfondo e primo piano. Se *c* è un numero compreso tra 0 e 63, l'enunciato PALETTE 0,*c* seleziona il colore *c* come sfondo e PALETTE 1,*c* seleziona il colore *c* come primo piano.

- Modalità SCREEN 3 (risoluzione grafica 720x348, 80 caratteri per riga)

  monitor monocromatici: questa modalità può essere usata solo se si dispone di una scheda Hercules; è simile al modo SCREEN 2, ma l'enunciato PALETTE non è supportato.

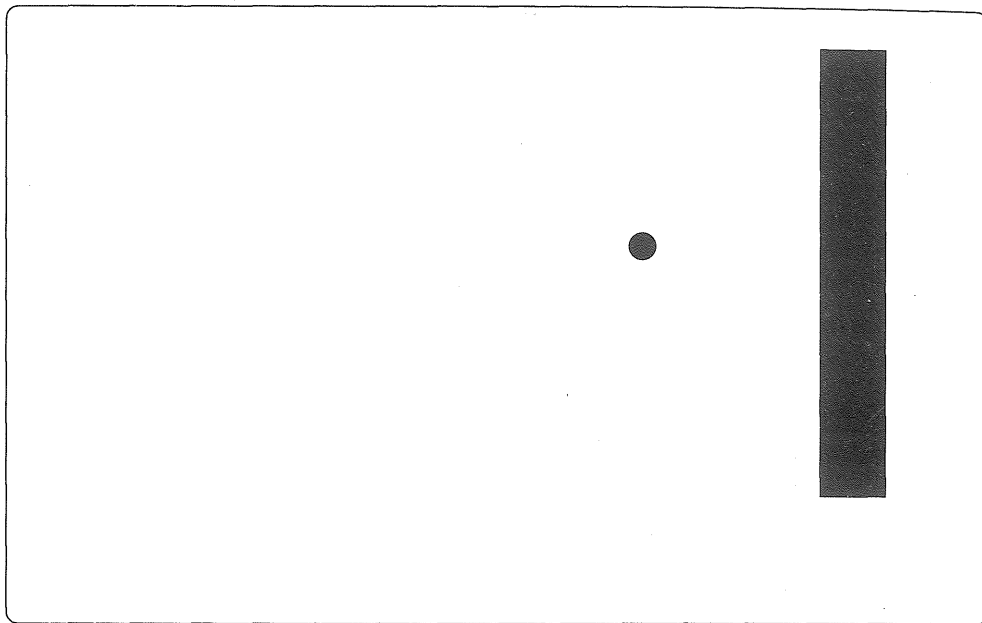



Figura 8.18 *La palla e la barriera*

- Modalità SCREEN 7 (risoluzione grafica 320x200, 40 caratteri per riga)

  monitor EGM o RGB: è disponibile una sola palette con 16 colori, numerati da 0 a 15. Inizialmente, al barattolo *m* viene assegnato il colore *m* della Tabella 8.5. Tuttavia, l'enunciato PALETTE può essere usato per assegnare uno qualsiasi dei 16 colori principali a un qualunque barattolo. Il comando COLOR *f,b* specifica che il colore nel barattolo *f* della palette deve essere usato per il primo piano (cioè il colore usato per il testo e per i grafici) e che il colore nel barattolo *b* deve essere usato per lo sfondo. Il parametro *m* negli enunciati grafici PSET, LINE, CIRCLE, DRAW e PAINT può essere compreso tra 0 e 15. Quando si impartisce un comando GET per catturare una porzione rettangolare dello schermo in un array, il secondo punto della procedura da seguire per determinare la dimensione dell'array deve essere cambiato nel modo seguente: il numero 2 deve essere cambiato in 1 e *v* deve essere sostituito da 4**v*.

Programma 8.19 *Impostazione di una nuova palette in modo SCREEN 1*

```
REM Nuovi colori per la palette 0 con monitor RGB e EGA  [8-19]
SCREEN 1, 0
COLOR , 0       'Seleziona la palette 0
PALETTE 0, 9    'Blu chiaro nel barattolo 0 (sfondo)
PALETTE 1, 14   'Giallo nel barattolo 1
PALETTE 2, 5    'Magenta nel barattolo 2
PALETTE 3, 10   'Verde chiaro nel barattolo 3
```

Programma 8.20 *Modifica dei colori in SCREEN 2*

```
REM Nuovi colori in SCREEN 2 con monitor RGB e EGA   [8-20]
SCREEN 2, 0
PALETTE 0, 9    'Colore di sfondo blu chiaro
PALETTE 1, 0    'Colore di primo piano nero
```

- Modalità SCREEN 8 (risoluzione grafica 640x200, 80 caratteri per riga)

  monitor EGM o RGB: questa modalità è analoga a SCREEN 7.

- Modalità SCREEN 9 (risoluzione 640x350, 80 caratteri per riga)

  monitor EGM: è disponibile una singola palette con 16 barattoli, numerati da 0 a 15. Inizialmente, al barattolo *m* viene assegnato il numero di colore che più si avvicina ai colori della Tabella 8.5. Gli enunciati COLOR, PALETTE e GET operano come nelle modalità SCREEN 7 e 8, ad eccezione del fatto che sono disponibili 64 colori che possono essere assegnati ai 16 barattoli. Tuttavia, se la scheda EGA contiene solo 64K di memoria, la palette consiste solamente di quattro barattoli, numerati da 0 a 3. Il parametro *m* negli enunciati grafici PSET, LINE, CIRCLE, DRAW e PAINT può essere compreso tra 0 e 3 o 0 e 15, a seconda della quantità di memoria disponibile.

- Modalità SCREEN 10 (risoluzione grafica 640x350, 80 caratteri per riga)

  monitor monocromatico: è disponibile una sola palette con quattro barattoli. Si possono assegnare 9 pseudo-colori (si veda la Tabella 8.9) a questi barattoli tramite il comando PALETTE. Se *m* è un numero compreso tra 0 e 3 e *c* un numero compreso tra 0 e 8, l'enunciato PALETTE *m,c* assegna lo pseudo-colore *c* al barattolo *m* della palette. Il comando COLOR *f,b* specifica che lo pseudo-colore nel barattolo *f* della palette deve essere usato per il primo piano (cioè, il colore da usare per il testo e per i grafici) e che lo pseudo-colore nel barattolo *b* della palette deve essere usato per lo sfondo. Per quanto riguarda il comando GET, questa modalità opera esattamente come i modi SCREEN 7 e 8. Il parametro *m* negli enunciati grafici PSET, LINE, CIRCLE, DRAW e PAINT può essere compreso tra 0 e 3.

Tabella 8.9 *Pseudo-colori disponibili con i monitor monocromatici*

0	nero
1	lampeggiante (nero su bianco)
2	lampeggiante (bianco su nero)
3	lampeggiante (nero su bianco intenso)
4	bianco
5	lampeggiante (bianco su bianco intenso)
6	lampeggiante (bianco intenso su nero)
7	lampeggiante (bianco intenso su bianco)
8	bianco intenso

USO DI UNA SCHEDA VGA

Se si dispone di una scheda e di un monitor VGA, le modalità SCREEN da 1 a 10 emulano le prestazioni di una scheda EGA. La VGA supporta inoltre tre nuovi modi SCREEN, 11, 12 e 13. Dato che l'enunciato COLOR non è disponibile in queste modalità, i colori vengono modificati tramite il comando PALETTE. Ognuno dei colori disponibili per riempire i barattoli ha la forma $c=(65536*b)+(245*g)+r$, dove b, g e r sono compresi tra 0 e 63 e rappresentano, rispettivamente, l'intensità di blu, verde e rosso.

- Modalità SCREEN 11 (risoluzione 640x480, 80 caratteri per riga): questa modalità è una versione avanzata di SCREEN 2. Si può specificare come colore di sfondo e di primo piano uno qualsiasi dei 256K di colori disponibili. Se *c* è un numero di colore valido, l'enunciato PALETTE 0,*c* imposta il colore di sfondo *c*, mentre il comando PALETTE 1,*c* imposta il colore di primo piano *c*. Il parametro *m* negli enunciati grafici PSET, LINE, CIRCLE, DRAW e PAINT può essere 0 o 1. Quando si impartisce un comando GET per catturare una porzione rettangolare dello schermo in un array, il numero 2 nel secondo punto della procedura da seguire per determinare la dimensione dell'array deve essere cambiato in 1.
- Modalità SCREEN 12 (risoluzione 640x480, 80 caratteri per riga): è disponibile una sola palette con 16 barattoli, numerati da 0 a 15. Inizialmente, al barattolo *m* viene assegnato il numero di colore che più si avvicina ai colori della Tabella 8.5. Tuttavia, l'enunciato PALETTE può essere usato per assegnare uno qualsiasi dei 256 colori disponibili a un qualunque barattolo. Se *c* è un numero di colore valido, l'enunciato PALETTE *j*,*c* riempie il barattolo *j* con il colore *c*. I barattoli 0 e 15 contengono i colori di default per lo sfondo e il primo piano. Il parametro *m* negli enunciati grafici PSET, LINE, CIRCLE, DRAW e PAINT può essere compreso tra 0 e 15. Quando si impartisce un comando GET per catturare una porzione rettangolare dello schermo in un array, il secondo punto della procedura da seguire per determinare la dimensione dell'array deve essere cambiato nel modo seguente: il numero 2 deve essere cambiato in 1 e *v* deve essere sostituito da $4*v$.
- Modalità SCREEN 13 (risoluzione 320x200, 40 caratteri per riga): è disponibile una singola palette con 256 barattoli, numerati da 0 a 255. Ai 256 barattoli si può assegnare un colore qualsiasi di quelli disponibili. Se *c* è un numero di colore valido, l'enunciato PALETTE *j*,*c* riempie il barattolo *j* con il colore *c*. Il parametro *m* negli enunciati grafici PSET, LINE, CIRCLE, DRAW e PAINT può essere compreso tra 0 e 255. Quando si impartisce un comando GET per catturare una porzione rettangolare dello schermo in un array, il numero 2 nel secondo punto della procedura da seguire per determinare la dimensione dell'array deve essere cambiato in 8.

SISTEMA DI COORDINATE DEFINITO DALL'UTENTE

Le coordinate discusse all'inizio di questo capitolo sono conosciute come coordinate fisiche. L'uso delle coordinate fisiche comporta due svantaggi. Innanzi tutto, le coordinate di un determinato punto sullo schermo dipendono dalla modalità SCREEN utilizzata. Ad esempio, il centro dello schermo ha coordinate (160,100) in modo SCREEN 1, (320,100) in SCREEN 2 e (320,175) in SCREEN 9. In secondo luogo, i sistemi di coordinate risultano difficili da usare in molte applicazioni, come quelle che devono visualizzare dei dati con grafici a barre o mostrare delle rappresentazioni grafiche delle funzioni matematiche. Si possono eliminare queste limitazioni usando il comando WINDOW, che consente di impostare un proprio sistema di coordinate. L'enunciato

```
WINDOW (x1,y1)-(x2,y2)
```

imposta un sistema di coordinate standard per lo schermo. In questo sistema, le coordinate x si estendono verso destra tra *x1* e *x2* e le coordinate y si estendono verso l'alto tra *y1* e *y2*, come mostrato in Figura 8.19a. L'enunciato

```
WINDOW SCREEN (x1,y1)-(x2,y2)
```

racchiude lo schermo in un sistema di coordinate non standard. In questo sistema, le coordinate x si estendono verso destra tra *x1* e *x2* e le coordinate y si estendono verso il basso tra *y1* e *y2*, come mostrato in Figura 8.19b. Questo sistema di coordinate definito dall'utente è conosciuto come *sistema di coordinate naturale.* Dopo aver eseguito un enunciato di questo tipo, i comandi PSET, PRESET, LINE, CIRCLE, GET e PUT utilizzano le nuove coordinate. Per ritornare al sistema di coordinate fisiche, è sufficiente impartire il comando WINDOW senza la parola chiave SCREEN e senza specificare nessuna coordinata. Inoltre, il sistema di coordinate fisiche viene automaticamente ripristinato quando si cambia la modalità SCREEN.

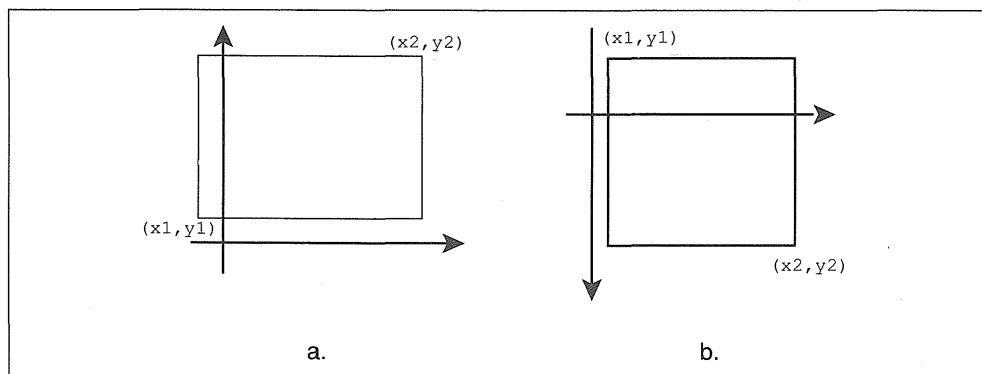



Figura 8.19 *I sistemi di coordinate specificati dall'enunciato WINDOW*

L'enunciato WINDOW non agisce sulla dimensione e la posizione dei caratteri di testo, sul comando DRAW o sulla scala di uno stile di linea o di un motivo di riempimento. Inoltre, il comando WINDOW imposta l'ultimo punto indirizzato al centro dello schermo. La funzione PMAP (descritta nell'Appendice G) consente di convertire le coordinate fisiche in coordinate naturali, e viceversa.

Il Programma 8.21 specifica un sistema di coordinate standard in cui le coordinate x sono comprese tra -5 e 5 e le coordinate y tra -5 e 100. Il risultato di questo programma è riportato in Figura 8.20.

Il Programma 8.22 specifica un sistema di coordinate non standard in cui le coordinate x sono comprese tra -200 e 2000 e le coordinate y tra -500 e 500. Il risultato di questo programma è riportato in Figura 8.21.

Ogni volta che si specifica un sistema di coordinate naturale in un programma, il sistema determina come devono essere inseriti i punti prima che venga impartito un altro enunciato WINDOW. Modificando il sistema di coordinate, è possibile ingrandire, rimpicciolire e spostare delle figure in diverse parti dello schermo. Il programma 8.23 genera la Figura 8.22.

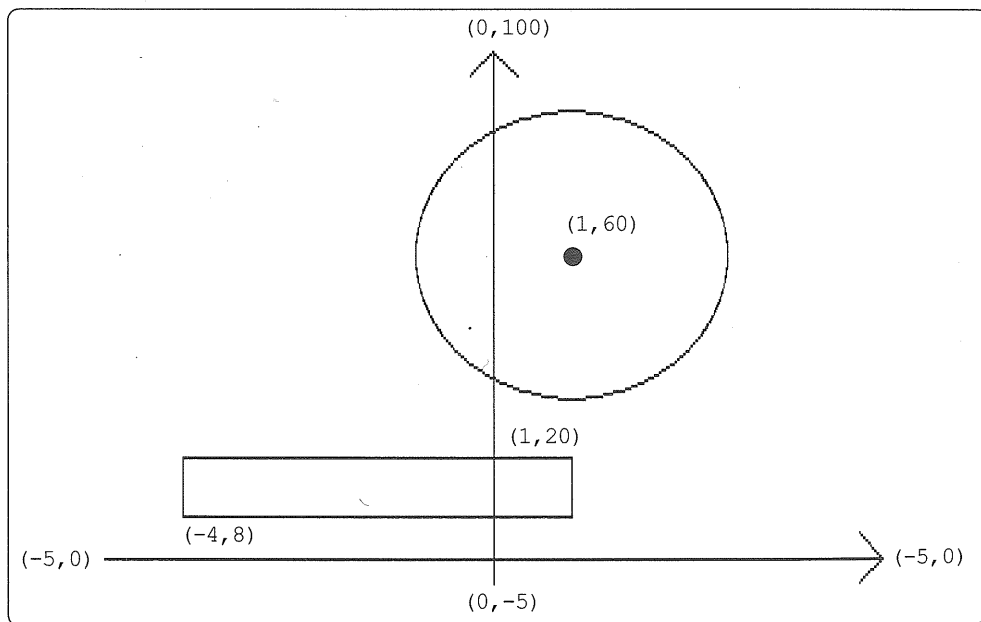



Figura 8.20 *Il risultato del Programma 8.21*

Programma 8.21 *Uso del comando WINDOW*

```
REM Uso dell'enunciato WINDOW [8-21]
SCREEN 2, 0
WINDOW (-5, -5)-(5, 100)
LINE (0, -5)-(0, 100)
LINE (0, 100)-(.3, 95)
LINE (0, 100)-(-.3, 95)
Traccia l'asse x e una freccia
LINE (-5, 0)-(5, 0)
LINE (5, 0)-(4.7, 5)
LINE (5, 0)-(4.7, -5)
'Traccia un cerchio e un rettangolo
CIRCLE (1, 60), 2
LINE (-4, 8)-(1, 20), , B
END
```

Programma 8.22 *Uso del comando WINDOW SCREEN*

```
REM Uso dell'enunciato WINDOW SCREEN [8-22]
SCREEN 2, 0
WINDOW SCREEN (-200, -500)-(2000, 500)
'Traccia l'asse y e una freccia
LINE (0, -500)-(0, 450) 'Lascia dello spazio per un messaggio
LINE (0, 450)-(70, 400)
LINE (0, 450)-(-70, 400)
'Traccia l'asse x e una freccia
LINE (-200, 0)-(2000, 0)
LINE (2000, 0)-(1930, 50)
LINE (2000, 0)-(1930, -50)
'Traccia un cerchio e una linea
CIRCLE (1100, 100), 400
LINE (100, -100)-(900, -400)
END
```

Programma 8.23 *Uso del comando WINDOW per ingrandire, rimpicciolire e spostare delle figure*

```
REM Traccia una figura in varie posizioni e dimensioni  [8-23]
SCREEN 1, 0
CALL DrawFigure
WINDOW SCREEN (0, 0)-(900, 600)
CALL DrawFigure
WINDOW SCREEN (-50, 50)-(190, 190)
CALL DrawFigure
END
```

```
SUB DrawFigure
  CIRCLE (160, 80), 20
  PAINT (160, 80), 2, 3
  LINE (160, 97)-(160, 150)
  LINE (160, 120)-(190, 110)
  LINE (160, 120)-(130, 110)
  LINE (160, 150)-(190, 180)
  LINE (160, 150)-(130, 180)
END SUB
```

IL COMANDO VIEW

QBasic consente di riservare una porzione rettangolare dello schermo e specificare che tutti i grafici successivi devono apparire solo in quella porzione. Una regione rettangolare di questo tipo è denominata viewport. Con questo sistema è molto semplice ingrandire o rimpicciolire delle porzioni rettangolari dello schermo, ed inserire un'immagine scalata in un punto qualsiasi. Si consideri la porzione rettangolare di uno schermo in modalità grafica a media risoluzione che abbia, come coordinate dei due angoli opposti, $(x1,y1)$ e $(x2,y2)$. L'enunciato

```
VIEW (x1,y1)-(x2,y2)
```

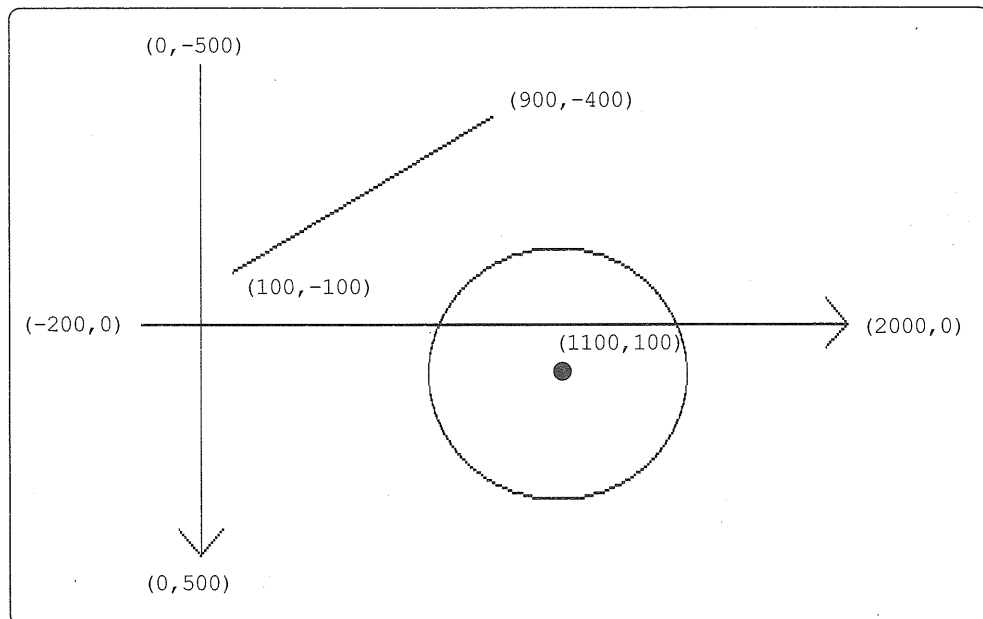



Figura 8.21 *Il risultato del Programma 8.22*

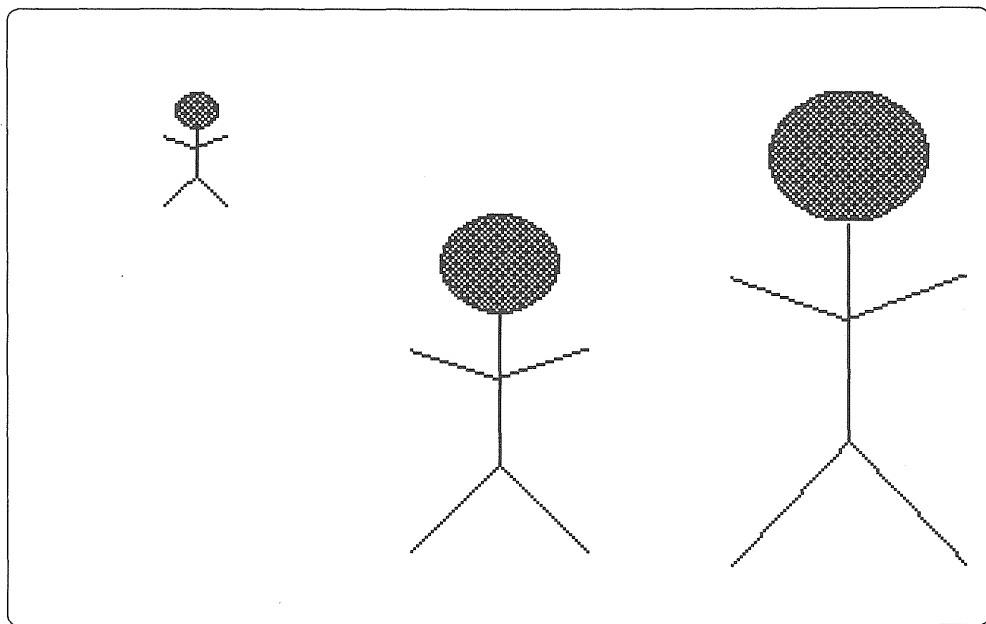



Figura 8.22 *Il risultato del Programma 8.23*

indica al programma di visualizzare nella regione rettangolare specificati tutti i grafici tracciati da PSET, PRESET, LINE e CIRCLE. In questo modo, la regione rettangolare diventa una viewport e funziona come un secondo schermo all'interno nello schermo fisico. Si possono aggiungere dei colori alla viewport a i suoi bordi tramite l'enunciato

```
VIEW (x1,y1)-(x2,y2), v, b
```

Questo comando assegna il colore del barattolo *v* all'interno della viewport e il colore del barattolo *b* ai suoi bordi. Se *v* o *b* viene omesso, alla viewport viene assegnato il colore di sfondo.

Il Programma 8.24 mostra l'effetto di alcuni comandi VIEW differenti. Si noti che quando il programma si ferma per la prima volta, l'output appare come quello riportato in Figura 8.23a. Dopo aver premuto Invio, viene eseguito il primo enunciato VIEW che definisce una viewport i cui angoli opposti hanno le coordinate (100,75) e (639,199) dello schermo fisico. Dato che come parametro relativo al bordo è stato specificato 1, il bordo appare come mostrato in Figura 8.23b. Dopo aver premuto Invio una seconda volta, viene eseguita la seconda coppia di enunciati LINE e CIRCLE. Si noti che questo sono uguali ai primi due comandi LINE e CIRCLE, ma che generano l'output riportato in Figura 8.23c. Ciò si verifica poiché il comando VIEW ha definito le coordinate dello schermo fisico (100,75)

come coordinate (0,0) della nuova viewport. In questo caso, si dice che la seconda immagine è stata 'tagliata' poiché non può essere contenuta nella viewport. Si noti che il comando VIEW e gli enunciati grafici successivi non influiscono sulle immagini già presenti sullo schermo.

Una volta definita, una viewport rispetta il sistema di coordinate naturali definito dall'istruzione WINDOW. Il Programma 8.25 traccia due volte una linea: una dopo aver impostato delle coordinate naturali tramite l'enunciato WINDOW, e l'altra in una piccola viewport (si veda la Figura 8.24). Si noti che i limiti della viewport vengono specificati in coordinate assolute, ma che il grafico è stato tracciato come se la viewport occupasse l'intero schermo. Ciò accade perché gli effetti dell'enunciato WINDOW restano attivi e vengono trasferiti nella nuova viewport. Il grafico appare allungato perché le coordinate della viewport impostate da VIEW non hanno lo stesso rapporto di forma dello schermo. Se il grafico fosse stato disegnato senza un comando WINDOW, la viewport avrebbe semplicemente tagliato le porzioni del grafico in eccesso.

Quando un comando VIEW è seguito da un altro enunciato VIEW, resta attiva solo la seconda viewport. Ciò significa che tutti gli enunciati grafici successivi verranno indirizzati alla seconda viewport. Il Programma 8.26 usa l'enunciato VIEW insieme al comando WINDOW per modificare la dimensione di una stella. L'output di questo programma è riportato in Figura 8.25.

Programma 8.24 *Gli effetti del comando VIEW*

```
REM Traccia un cerchio e un rettangolo in viewport differenti [8-24]
SCREEN 2
LINE (0, 0)-(639, 199)
CIRCLE (320, 100), 175
LOCATE 2, 40
INPUT "Premere Invio per impostare la viewport...", dummy$
VIEW (100, 75)-(639, 199), , 1
LOCATE 2, 40
INPUT "Premere Invio per tracciare la stessa immagine...", dummy$
LOCATE 2, 40: PRINT SPACE$(39)    ' Scrive 39 spazi
LINE (0, 0)-(639, 199)
CIRCLE (320, 100), 175
END
```

Se il comando VIEW *(x1,y1)-(x2,y2), v, b* venisse sostituito dall'enunciato

```
VIEW SCREEN (x1,y1)-(x2,y2), v, b
```

non si verificherebbe nessun riposizionamento o ridimensionamento. Quando viene disegnata una figura, invece, viene visualizzata solo la

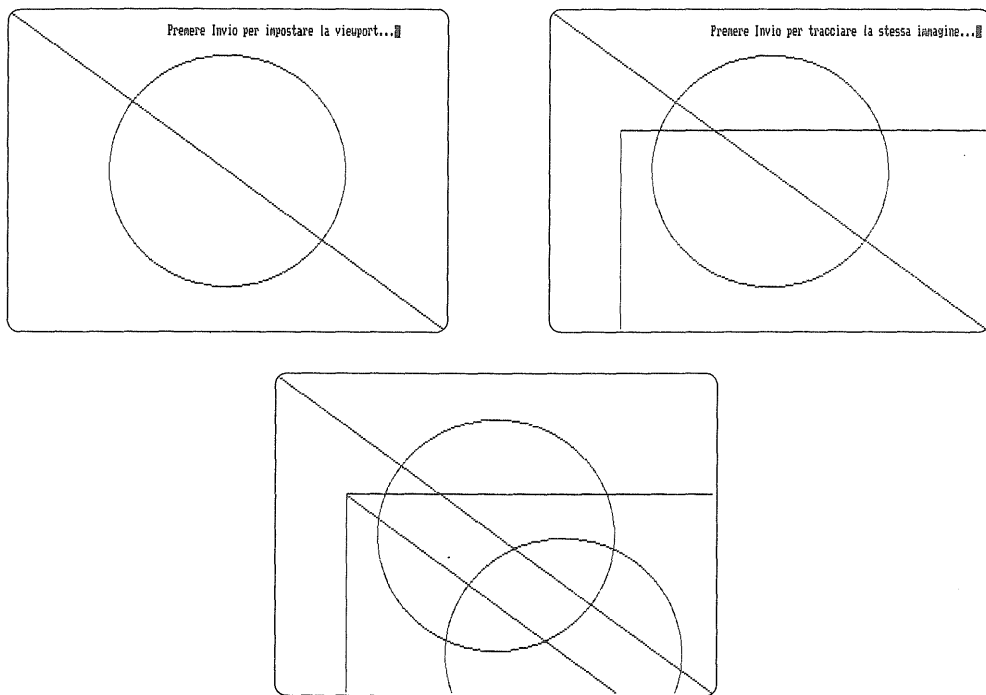



Figura 8.23 *Il risultato del Programma 8.24*

porzione all'interno della viewport. Si consideri il cubo tracciato dal Programma 8.2. Il Programma 8.27 disegna una porzione del cubo eseguendo un comando VIEW SCREEN prima dell'operazione. L'output del programma è riportato in Figura 8.26.

Programma 8.25 *Uso del comando WINDOW insieme a VIEW*

```
REM Disegna due volte la stessa immagine, ma la seconda volta
REM la scala con il comando VIEW  [8-25]
SCREEN 2
WINDOW (-.5, -4)-(7.5, 16)
CALL LineChart                   'Prima rappresentazione
VIEW (130, 117)-(550, 180), , 1
CALL LineChart                   'Seconda rappresentazione
DATA 0, 14, 7, 9, 6, 12
END

SUB LineChart
  CLS
  RESTORE
  LINE (0, 0)-(7, 0)     'x-axis
  LINE (0, 14)-(0, 0)    'y-axis
```

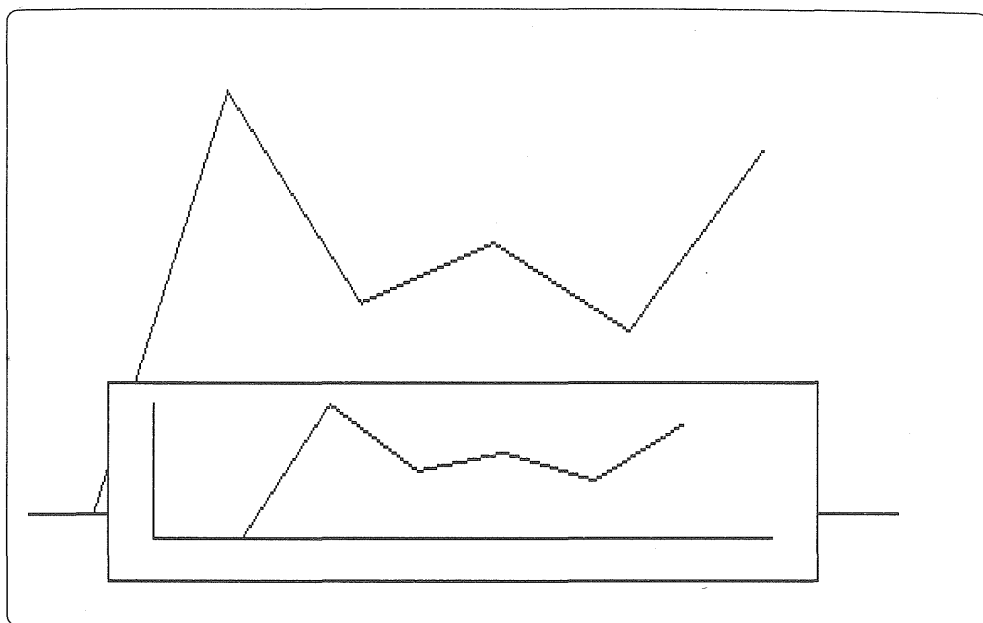



Figura 8.24 *Output del Programma 8.25*

```
  READ a
  PSET (1, a)
  FOR m = 2 TO 6
    READ a
    LINE -(m, a)
  NEXT m
END SUB
```

Programma 8.26 *Esempi d'uso del comando VIEW*

```
REM Disegna quattro stelle  [8-26]
SCREEN 1, 0
COLOR , 1
WINDOW (-8, -8)-(8, 8)
VIEW (1, 1)-(159, 99), , 3
CALL DrawStar(1)
VIEW (161, 101)-(240, 149), , 1
CALL DrawStar(2)
VIEW (242, 151)-(282, 176), , 2
CALL DrawStar(3)
VIEW (284, 178)-(304, 190), , 3
CALL DrawStar(1)
END
```

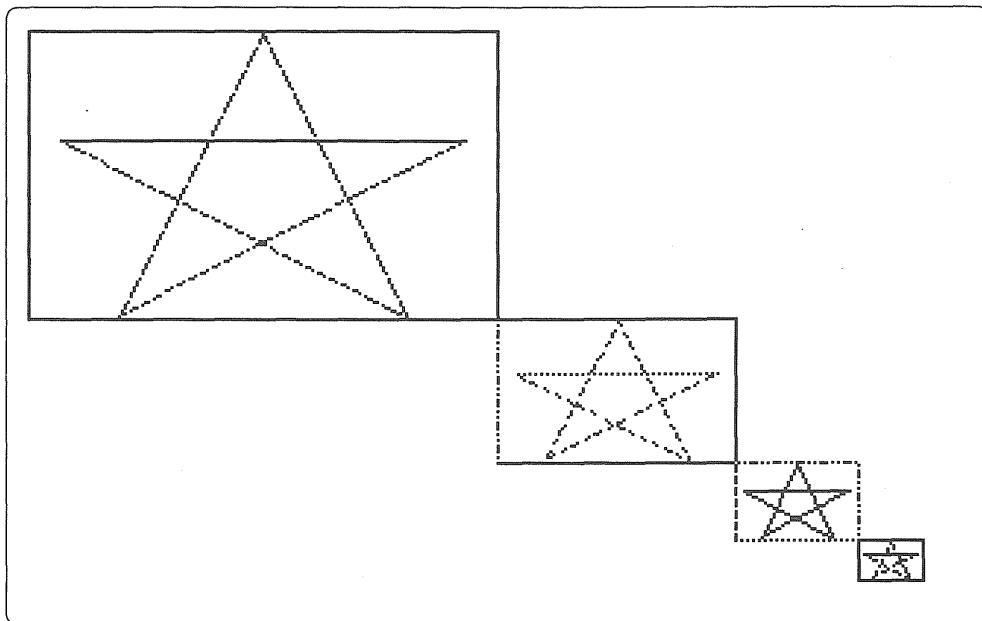



Figura 8.25 *Il risultato del Programma 8.26*

```
SUB DrawStar (starColor)
  PSET (0, 8)
  LINE -(5, -8), starColor
  LINE -(-7, 2), starColor
  LINE -(7, 2), starColor
  LINE -(-5, -8), starColor
  LINE -(0, 8), starColor
END SUB
```

Programma 8.27 *Uso del comando VIEW SCREEN*

```
REM Taglia il disegno di un cubo  [8-27]
SCREEN 1, 0
VIEW SCREEN (135, 35)-(195, 105), , 3
DRAW "L60 U60 R60 D60 E20 U60 G20 E20 L60 G20"
END
```

L'enunciato VIEW non influisce sulla dimensione e la posizione dei caratteri di testo, sul comando DRAW o sulla scala di uno stile di linea o di un motivo di riempimento. Se viene eseguito un comando CLS mentre è attivo un enunciato VIEW, viene cancellato solo il contenuto della viewport. Ciò vale sia per il testo che per i grafici. Per cancellare l'intero schermo senza alterare gli

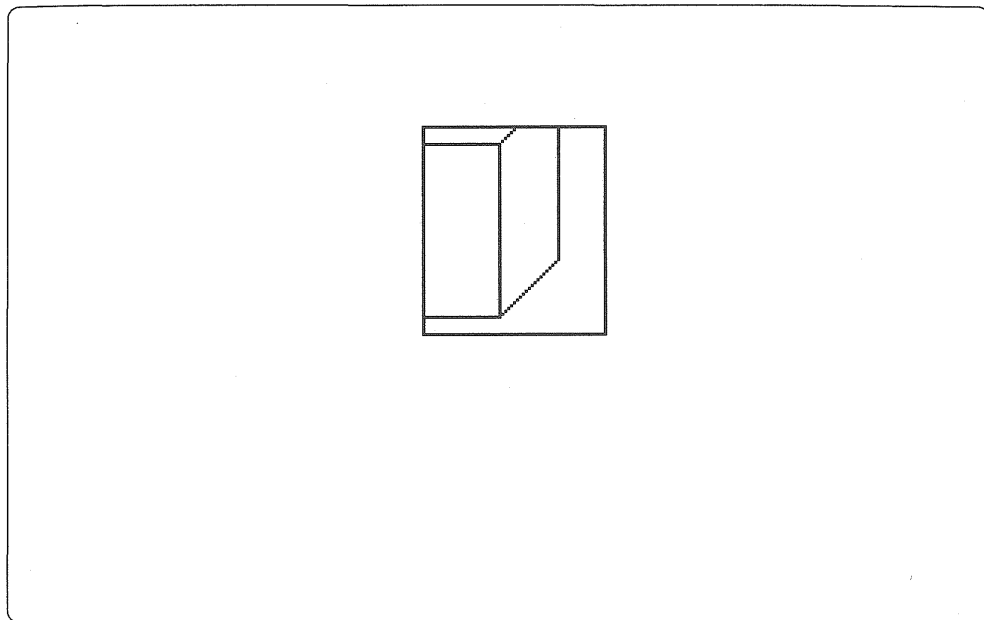



Figura 8.26 *Il risultato del Programma 8.27*

enunciati WINDOW e VIEW, si impartisca il comando PRINT CHR$(12). Per cancellare l'intero schermo e disattivare la viewport, si usi l'istruzione VIEW: CLS.

SUONO

QBasic consente di generare dei suoni dall'altoparlante del PC tramite il comando PLAY. A questo scopo, si devono definire le note e la loro lunghezza, il tempo della composizione, e si deve decidere se il programma deve fermarsi o meno durante la riproduzione delle note.

Una tastiera di un piano consiste di 88 tasti; dal computer si possono riprodurre le note associate a 84 di questi tasti. Nella Figura 8.27, questi 84 tasti sono raggruppati in sette ottave etichettate da 0 a 6.

Ogni nota viene identificata da un'ottava e da una lettera (da A a G). I diesis e i bemolle vengono definiti, rispettivamente, dai suffissi # (o +) e -. Un enunciato PLAY consiste della parola chiave PLAY seguita da una stringa

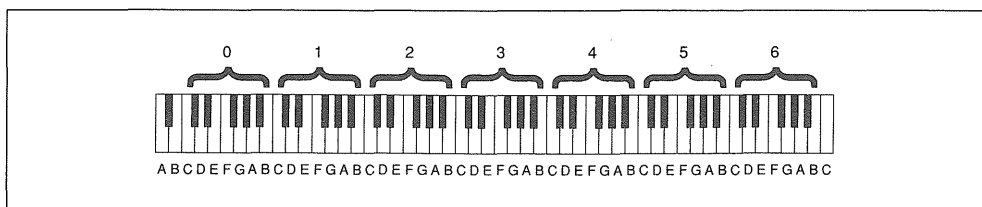



Figura 8.27 *La tastiera di un piano*

contenente le informazioni relative alle note da riprodurre. Ad esempio, l'enunciato

```
PLAY "O3 C"
```

riproduce la nota C dell'ottava 3. L'enunciato

```
PLAY "O2 DE O4 E#B-"
```

riproduce quattro note in sequenza, le prime due appartenenti all'ottava 2, e le altre due all'ottava 4. In generale, la lettera O seguita da un numero compreso tra 0 e 6 indica al programma di riprodurre tutte le note successive utilizzando l'ottava specificata. Anche gli enunciati PLAY successivi sono influenzati da questa specifica. Se non viene specificata un'ottava prima di impartire il primo comando PLAY, viene utilizzata l'ottava 4. L'ottava può essere incrementata o decrementata in qualsiasi momento inserendo, rispettivamente, i caratteri > o < nella stringa PLAY.

Il Programma 8.28 'trasforma' il computer in una tastiera elettronica. Durante l'esecuzione del programma, si possono riprodurre delle note premendo i tasti da A a G e tenendoli premuti per quanto si desidera. L'ottava può essere cambiate premendo, per un numero appropriato di volte, i tasti < o >. Per uscire dal programma si prema il tasto Q.

Il Programma 8.29 riproduce alcune note. Dato che non viene specificata nessuna ottava davanti al primo enunciato PLAY, le prime due note (E e F diesis) vengono riprodotte dall'ottava 4 (l'ottava di default). Le altre cinque note (C, D bemolle, E, E ed F diesis) fanno tutte parte della prima ottava. Le specifica O1 resta attiva per tutti gli enunciati PLAY successivi, fino a quando non viene specificata un'altra ottava. Si sarebbero potuti riunire i tre comandi PLAY nell'unico enunciato PLAY "EF#O1CD-EEF#". Il Programma 8.30 suona la scala in re minore.

Quando si usa l'enunciato PLAY si può scegliere tra due diverse modalità operative. La modalità MF indica al programma di riprodurre le note associate al comando PLAY prima di eseguire l'enunciato successivo. La modalità MB, invece, consente di memorizzare fino a 32 note in un'area di memoria temporanea e di riprodurle mentre l'esecuzione del programma continua. Queste due modalità vengono attivate tramite gli enunciati PLAY "MF" e PLAY "MB". La modalità operativa di default è MF.

Programma 8.28 *Una tastiera elettronica*

```
REM Una tastiera elettronica  [8-28]
CLS
PRINT "Tasti: A B C D E F G per suonare < o > per ";
PRINT "cambiare l'ottava, e Q per uscire."
a$ = ""
DO UNTIL a$ = "Q"
  SELECT CASE a$
    CASE "A", "B", "C", "D", "E", "F", "G", ">", "<", ""
       PLAY a$
    CASE ELSE
       LOCATE 3, 1: PRINT "Non è una nota!";
  END SELECT
  a$ = UCASE$(INPUT$(1))
  LOCATE 3, 1: PRINT SPACE$(18);     ' Scrive 18 spazi
LOOP
END
```

Programma 8.29 *Specifica di un'ottava*

```
REM Riproduce 7 note  [8-29]
PLAY "EF+"
PLAY "O1 CD-E"
PLAY "EF#"
END
```

Programma 8.30 *Riproduzione della scala in re minore*

```
REM Scala in Re minore  [8-30]
PLAY "D E F G A B- O5 C D"
END
```

Quando si esegue il Programma 8.31, appaiono le parole non appena inizia la riproduzione del motivo. Se la specifica MB nella terza riga venisse sostituita da MF, le parole non apparirebbero fino al termine della composizione.

Le note musicali standard possono essere intere (1/1), mezze (1/2), quarti (1/4), ottave (1/8), sedicesimi (1/16), trentaduesimi (1/32) e sessantaquattresimi (1/64). Il computer non solo è in grado di riprodurre tutte queste lunghezze, ma può generare note di lunghezza $1/n$ dove n può essere compreso tra 1 e 64.

Quando una lettera da A a G in un enunciato PLAY è seguita dal numero n, quella nota avrà una lunghezza di $1/n$. Per esempio, l'enunciato

```
PLAY "C2 C1 C25"
```

riproduce tre volte la nota C; la prima volta come mezza nota, la seconda come nota intera, e la terza come nota di 1/25.

La lunghezza delle note può anche essere specificata dalla lettera L seguita da un numero compreso tra 1 e 64, che indica al programma di utilizzare quella lunghezza per tutte le note successive fino a quando non viene specificato diversamente. Anche gli enunciati PLAY successivi vengono influenzati dalla lunghezza specificata. Se non viene specificata nessuna lunghezza davanti al primo comando PLAY, le note vengono riprodotte con una lunghezza pari a un quarto. Ad esempio, l'enunciato

```
PLAY "CC8 L16 CCC L1 CCC2"
```

riproduce otto volte la nota C: la prima come quarto, la seconda come ottavo, quindi tre volte come sedicesimo, due volte come intero e l'ultima come mezzo.

Programma 8.31 *Dimostrazione della modalità MB*

```
REM Motivo iniziale di Happy Birthday  [8-31]
CLS
PLAY "MB"
PLAY "CCDCFE"
PRINT "Happy Birthday to You"
END
```

Per inserire il valore di una variabile numerica in un enunciato PLAY, è necessario convertire il valore della variabile in stringa tramite la funzione STR$ ed inserire la stringa risultante nella posizione appropriata usando l'operatore +. La Tabella 8.10 mostra alcuni esempi dell'enunciato PLAY con le variabili. Si possono usare delle variabili al posto di altri parametri di un enunciato PLAY in modo analogo. Il Programma 8.32 usa una variabile di un ciclo FOR/NEXT per riprodurre la nota C con 64 lunghezze differenti.

Tabella 8.10 *Uso di variabili nell'enunciato PLAY*

Enunciato	Formato equivalente con le variabili
`PLAY "C4"`	`a=4: PLAY "C"+STR$(a)`
`PLAY "L2 D"`	`lungh=2: PLAY "L"+STR$(lungh)+"D"`

Si si aggiunge la lettera P seguita da un numero n compreso tra 1 e 64, so ottiene una pausa della durata di $1/n$. Ad esempio, l'enunciato

```
PLAY "C P2 P16 C"
```

suona tre volte la nota C, con delle pause di un mezzo e di un sedicesimo.

In notazione musicale standard, un punto dopo una nota o una pausa indica una lunghezza pari a una volta e mezza la lunghezza normale. In un enunciato PLAY, un punto ha lo stesso significato. Per esempio, l'enunciato

```
PLAY "C C. C8. C.. L15 P4. C2. C."
```

suona sei volte la nota C con lunghezza di 1/4, 3/8, 3/16, 9/16, 3/4 e 1/10. È inoltre presente una pausa di 3/8 tra la quarta e la quinta nota.

Programma 8.32 *Uso di una variabile in un enunciato PLAY*

```
REM Modifica della lunghezza delle note [8-32]
CLS
LOCATE 12, 1
PRINT "Questa è una nota intera."
PLAY "MF O3 C1"
FOR n = 2 TO 64
  LOCATE 12, 10
  PRINT " 1 /"; n; "nota."
  PLAY "L" + STR$(n) + "C"
NEXT n
END
```

In notazione musicale standard, un punto sopra o sotto una nota indica che la nota deve essere breve e diesis, con una pausa tra la nota e quella successiva. Questo formato è chiamato *staccato*. Una linea curva sopra o sotto alcune note significa che queste devono essere riprodotte senza pause tra una nota e l'altra. Questo formato è chiamato *legato*. Si può indicare lo staccato e il legato in un enunciato PLAY utilizzando, rispettivamente, le coppie di lettere MS e ML. La coppia di lettere MN indica il modo standard. Le lettere MS all'interno di un enunciato PLAY indicano al programma di riprodurre tutte le note successive in staccato fino a quando non viene incontrata una coppia di lettere MN o ML. Le stesse considerazioni valgono per ML.

Il Programma 8.33 suona la prima parte del motivo "Happy Birthday" in modo staccato, la seconda in modo normale e la terza in modo legato. Si noti che, in modo legato, le prime due note C si fondono in una nota sola. Il Programma 8.34 suona l'inizio della canzone "Frere Jacques" usando la lunghezza delle note definita dall'utente.

La velocità o il tempo di una composizione viene specificata con parole in italiano. Alcuni dei tempi più comuni sono riportati nella Tabella 8.11. In un enunciato PLAY, si può specificare il tempo di una composizione posponendo alla lettera T un numero *n* compreso tra 32 e 255; ciò indica a QBasic di suonare tutte le note successive alla velocità di *n* quarti di nota per minuto fino a quando non viene specificato un altro tempo. Se non si specifica un tempo davanti al primo enunciato PLAY, viene utilizzata

l'impostazione di default di 120 quarti di nota per minuto fino a quando viene specificato diversamente. Ad esempio, l'enunciato

```
PLAY "C T60 C"
```

suona due volte la nota C, la prima per mezzo secondo e l'altra per un secondo. Il Programma 8.35 suona la scala in do maggiore con ciascuno dei tempi riportati nella Tabella 8.11.

Programma 8.33 *Modo staccato, normale e legato*

```
REM Happy Birthday in tre stili  [8-33]
b$ = "CCDCFE"
PLAY "MS" + b$
PLAY "MN" + b$
PLAY "ML" + b$
END
```

In questa discussione, ognuna delle 84 note disponibili per gli enunciati PLAY è stata identificata da una combinazione di un'ottava (da 0 a 6) e da una lettera (da A a G). Queste note possono anche essere identificate dalla lettera N seguita da uno dei numeri compresi tra 1 e 84, come mostrato nella Tabella 8.2.

Si può usare la combinazione N0 per indicare una pausa (questa combinazione è utile poiché, a differenza della combinazione Pn, N0 usa la lunghezza delle note di default). Ad esempio, l'enunciato

```
PLAY "N37 N0 N38"
```

suona la nota C, effettua una pausa, e quindi la nota C diesis. Il Programma 8.36 usa i numeri per identificare le note.

QBasic dispone di un altro enunciato per riprodurre delle note. Il comando

```
SOUND f, d
```

riproduce un suono con frequenza di *f* hertz con una durata di *d* * 0,055 secondi. Benché la frequenza possa essere compresa tra 37 e 32767 hertz, l'orecchio umano può recepire solamente una frequenza di circa 20.000 hertz. Il Programma 8.37 usa l'enunciato SOUND per creare degli effetti sonori speciali.

Programma 8.34 *Uso della funzione STR$ nell'enunciato PLAY*

```
REM Inizio del motivo Fre`re Jacques  [8-34]
CLS
INPUT "Lunghezza delle note (1-64)"; n
f$ = "CDEC"
PLAY "L" + STR$(n) + f$ + f$ + "EFG P" + STR$(n) + "EFG"
END
```

Tabella 8.11 *Tempi*

Tempo	Numero approssimativo di quarti di nota per minuto
Largo	50
Adagio	70
Andante	90
Moderato	110
Allegro	130
Vivace	150
Presto	170

In questo capitolo sono state esaminate le funzioni relative alla grafica e al suono. Negli ultimi due capitoli di questo libro, saranno trattati gli strumenti disponibili per calcoli matematici e finanziari, e verranno proposti alcuni esempi.

Programma 8.35 *Uso di tempi differenti*

```
REM Scala di Do  [8-35]
CLS
PLAY "MF"
FOR n = 50 TO 170 STEP 20
  PRINT n; "quarti di nota per minuto"
  PLAY "T" + STR$(n) + "O3 CDEFGAB O4 C"
NEXT n
END
```

Tabella 8.12 *Il numero associato a ciascuna nota*

Ottava0		Ottava1		Ottava2		Ottava3		Ottava4		Ottava5		Ottava6	
Nota	Num.	Nota	Num.	Nota	Num.	Nota	Num.	Nota	Num.	Nota	Num.	Nota	Num.
C	1	C	13	C	25	C	37	C	49	C	61	C	73
C+	2	C+	14	C+	26	C+	38	C+	50	C+	62	C+	74
D	3	D	15	D	27	D	39	D	51	D	63	D	75
D+	4	D+	16	D+	28	D+	40	D+	52	D+	64	D+	76
E	5	E	17	E	29	E	41	E	53	E	65	E	77
F	6	F	18	F	30	F	42	F	54	F	66	F	78
F+	7	F+	19	F+	31	F+	43	F+	55	F+	67	F+	79
G	8	G	20	G	32	G	44	G	56	G	68	G	80
G+	9	G+	21	G+	33	G+	45	G+	57	G+	69	G+	81
A	10	A	22	A	34	A	46	A	58	A	70	A	82
A+	11	A+	23	A+	35	A+	47	A+	59	A+	71	A+	83
B	12	B	24	B	36	B	48	B	60	B	72	B	84

Programma 8.36 *Riproduzione di un motivo usando i numeri*

```
REM Jingle Bells  [8-36]
CLS
PRINT "Jingle bells, Jingle bells"
PLAY "MF L8 N41 N41 L4 N41 L8 N41 N41 L4 N41"
PRINT "Jingle all the way"
PLAY "L8 N41 N44 N37. L16 N39 L4 N41 N0"
PRINT "Oh what fun it is to ride in a"
PLAY "L8 N42 N42 N42. L16 N42 L8 N42 N41 N41 L16 N41 N41"
PRINT "One horse open"
PLAY "L4 N44 N44 N42 N39"
PRINT "Sleigh"
PLAY "L1 N37"
END
```

Programma 8.37 *Effetti speciali con l'enunciato SOUND*

```
REM Lancio di una bomba, orologio, e sirena  [8-37]
t0 = TIMER
FOR i = 1 TO 500: NEXT i
loopsPerSec = 500 / (TIMER - t0)
'loopsPerSec = # of passes through loop to delay about 1 second
CLS
'Bomba
PRINT "Lancio di una bomba"
FOR n = 1000 TO 700 STEP -5
  SOUND n, 1
NEXT n
FOR n = 1 TO 700
  SOUND 50 * RND + 37, .0015
NEXT n
SLEEP 1
'Orologio
PRINT "Orologio"
FOR n = 1 TO 5
  SOUND 500, .1
  FOR i = 1 TO loopsPerSec * .4: NEXT i
  SOUND 2000, .1
  FOR i = 1 TO loopsPerSec * .4: NEXT i
NEXT n
SLEEP 1
'Sirena
PRINT "Sirena"
FOR n = 1 TO 5
  SOUND 1700, 5
  SOUND 1000, 5
NEXT n
```

CAPITOLO 9

PROGRAMMI MATEMATICI E SCIENTIFICI

Dopo aver esaminato le istruzioni relative alla grafica e al suono, ci si sposterà su argomenti un po' più complessi. QBasic dispone di alcuni strumenti matematici e scientifici che consentono di risolvere problemi sofisticati.

Dato che un programma matematico è limitato quasi esclusivamente a calcoli numerici (benché la grafica possa ricoprire un ruolo importante), le funzioni che verranno introdotte in questo capitolo consentono di gestire dati numerici. Saranno esaminate delle funzioni completamente nuove che non sono state neppure accennate nel corso di questo libro, e verranno mostrati alcuni esempi.

FUNZIONI MATEMATICHE INCORPORATE

Il Capitolo 4 ha presentato le funzioni aritmetiche incorporate ABS, FIX, INT e SGN. In questo capitolo verranno esaminate le funzioni trigonometriche ATN, COS, SIN e TAN, la funzione matematica SQR, la funzione esponenziale EXP e la funzione logaritmica LOG. Verrà fornita una spiegazione dettagliata per ciascuna di queste funzioni.

LA FUNZIONE DI RADICE QUADRATA

Per qualsiasi numero non negativo *x*, il valore di

```
SQR(x)
```

è il numero non negativo il cui quadrato è *x*. Ad esempio, SQR(25) restituisce 5 e SQR(1) restituisce 1. Dato che la radice quadrata della maggior parte dei numeri non fornisce un numero intero, bisogna prestare molta attenzione al tipo di variabile usato per memorizzare il risultato.

Equazione di secondo grado Le radici (soluzioni) dell'equazione di secondo grado

$$ax^2 + bx + c = 0$$

sono

```
(-b + SQR(b^2 - 4*a*c)) / 2*a
```

e

```
(-b - SQR(b^2 - 4*a*c)) / 2*a
```

purché $b^2 - 4ac$ sia >= 0.

Ipotenusa di un triangolo rettangolo La lunghezza dell'ipotenusa di un triangolo rettangolo i cui cateti hanno lunghezza *a* e *b*, è

```
SQR(a^2 + b^2)
```

FUNZIONI TRIGONOMETRICHE

Un cerchio di raggio 1 è chiamato cerchio unitario. La circonferenza di un cerchio unitario è uguale a 2*pi, dove pi è uguale, con una buona approssimazione, a 3,14592653589793. La Figura 9.1 mostra un cerchio unitario con un angolo il cui lato si estende lungo la parte positiva dell'asse x.

Gli angoli possono essere misurati sia in gradi che in radianti; quando si misurano in radianti, la dimensione dell'angolo equivale alla lunghezza dell'arco del cerchio unitario sotteso all'angolo. Ad esempio, dato che un angolo retto sottende un quarto del cerchio unitario, un angolo retto è uguale a (2*pi)/4 o pi/2 radianti. Un altro modo per ottenere la misura in radianti di un angolo è quello di moltiplicare il numero di gradi dell'angolo per pi/180. Ad esempio, dato che un angolo retto è di 90 gradi, la sua misura in radianti è uguale a 90*(pi/180) o pi/2.

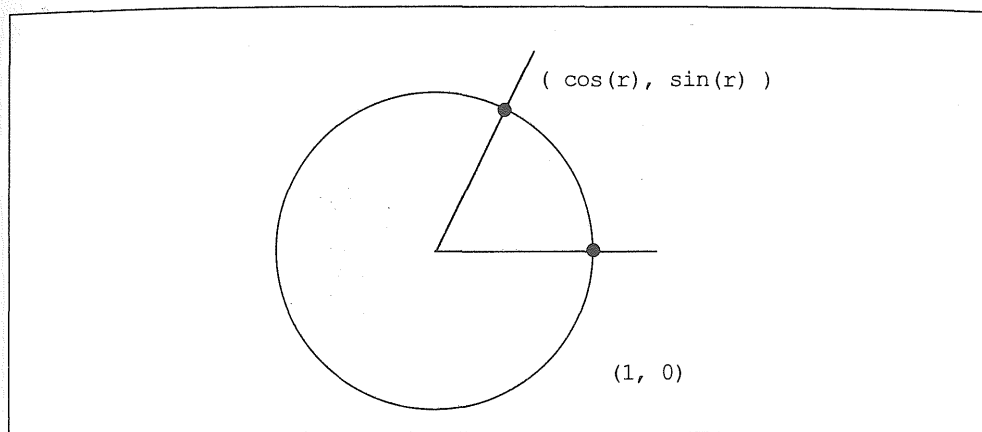



Figura 9.1 *Un angolo tracciato in un cerchio unitario*

Ognuna delle funzioni trigonometriche assegna un numero a un angolo. Si consideri un angolo di *r* radianti inserito in un cerchio unitario, come mostrato in Figura 9.1, che interseca il cerchio nel punto *P*. Il valore di COS(*r*) è la prima coordinata del punto *P* e il valore di SIN(*r*) è la seconda coordinata di *P*. Il valore di TAN(*r*) equivale a SIN(*r*)/COS(*r*). Queste tre funzioni calcolano, rispettivamente, il seno, il coseno e la tangente di un angolo. Se *x* è un numero qualsiasi, il valore di ATN(*x*) rappresenta un angolo in radianti la cui tangente è *x*. (Se risulta necessario, si può ottenere una buona approssimazione di pi usando la formula 4*ATN(1)).

Misurazione L'altezza dell'albero mostrato in Figura 9.2 è *d**TAN(*r*).

Fenomeni periodici Le funzioni trigonometriche possono essere di aiuto nella creazione di modelli per avvenimenti naturali ciclici. Ad esempio, la temperatura dell'acqua di rubinetto a Dallas nel Texas nell'*ennesimo* giorno dell'anno, è di circa 59+(14*COS((*n*-208)*pi/183) gradi Fahrenheit. Il Programma 9.1 usa questa formula.

Moto di un oggetto Se una palla da golf viene colpita con un angolo di *r* radianti rispetto al terreno a una velocità di *s* piedi per secondo, la distanza percorsa dalla palla è uguale a *s*^2*COS(*r*)*SIN(*r*)/16. Si noti che questa formula presume una minima resistenza da parte del vento. Il Programma 9.2 utilizza la velocità di 170 piedi al secondo per calcolare la distanza percorsa da una palla da golf.

FUNZIONI ESPONENZIALI

Qualsiasi funzione nella forma

b^x

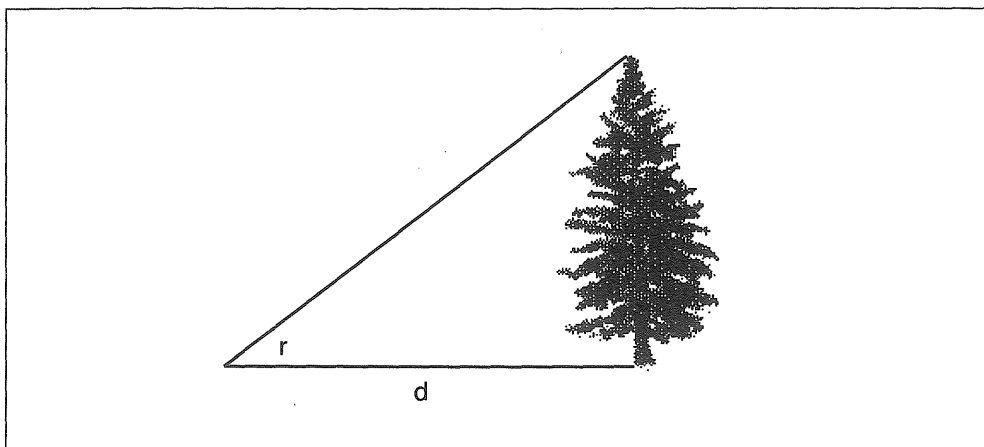



Figura 9.2 *Calcolo dell'altezza di un albero*

è chiamata funzione esponenziale. Il numero *b* è la base e il numero *x* l'esponente. Se la base ha valore e, dove e è uguale approssimativamente a 2,718281828459045 (lo stesso numero di cifre significative gestibili da una variabile a precisione doppia), si può usare la funzione EXP(*x*) al posto dell'espressione 2.718281828459045#^*x*. Il numero e è simile al numero pi nel senso che non si può scrivere il valore esatto, ma bisogna sempre utilizzare una forma approssimata. Ad esempio, l'enunciato

```
PRINT EXP(3)
```

calcola 3^e e restituisce il risultato 20,08554.

Programma 9.1 *Un esempio delle funzioni COS e ATN*

```
REM Previsione sulla temperatura dell'acqua di rubinetto [9-1]
CLS
INPUT "Giorno dell'anno (1-365)"; day%
pi = 4 * ATN(1)
t = 59 + 14 * COS((day% - 208) * pi / 183)
PRINT USING "La temperatura dell'acqua di rubinetto sarà ## gradi"; t
END

[Esecuzione]
Giorno dell'anno (1-365)? 180
La temperatura dell'acqua di rubinetto sarà 71 gradi
```

Curva normale In statistica, la *curva normale* viene usata per calcolare quante sono le probabilità che una misurazione appaia in un determinato intervallo. Per una popolazione con media *m* e deviazione standard *s* (un valore che indica quanto distanti dalla media tendono ad andare i diversi valori), la curva normale è il grafico della funzione (1/(*s**SQR(2*pi)))*EXP(-.5*((*x*-*m*)/*s*)^2).

Programma 9.2 *Uso delle funzioni trigonometriche con angoli dati in gradi*

```
REM Calcola la distanza percorsa da una palla da golf  [9-2]
REM che parte alla velocità di 170 piedi al secondo
CLS
PRINT "Con quale angolo rispetto al terreno è stata"
INPUT "colpita la palla da golf (0-90)"; degrees
pi = 4 * ATN(1)
radians = degrees * pi / 180
PRINT "Una palla che parte alla velocità di 170 piedi al secondo"
PRINT "a un angolo di"; degrees; "gradi percorre una distanza"
PRINT USING "di ### piedi"; 170 ^ 2 * COS(radians) * SIN(radians) / 16
END

[Esecuzione]
Con quale angolo rispetto al terreno è stata
colpita la palla da golf (0-90)? 45
Una palla che parte alla velocità di 170 piedi al secondo
a un angolo di 45 gradi percorre una distanza
di 903 piedi
```

Per misurazioni distribuite normalmente, la percentuale di misurazioni presente tra i valori *a* e *b* equivale all'area sotto alla curva normale, da $x=a$ a $x=b$. Una volta fornita la media e la deviazione standard di una popolazione, il Programma 9.3 trova l'area al di sotto della curva normale per calcolare le probabilità che un evento si verifichi in un determinato intervallo di valori.

Per dare un senso al risultato generato dal Programma 9.3, si supponga che i numeri inseriti rappresentino i dati relativi a un test effettuato in una classe di studenti. Il punteggio medio ottenuto è stato di 80 su 100. Circa due terzi degli studenti sottoposti al test hanno ottenuto un punteggio che non dista più di 10 punti dalla media (questa è la deviazione standard). Il risultato fornito dal programma indica che solo 1 studente su 22 ha ottenuto un punteggio maggiore o uguale a 95.

Interesse continuo Molte banche pagano degli interessi composti trimestralmente, mensilmente o giornalmente. Tuttavia, alcune banche calcolano gli interessi capitalizzati. Se si depositano *P* lire a un tasso di interesse *r* capitalizzato, il saldo del conto corrente dopo *t* anni sarà uguale a *P**EXP(*r***t*). Ad esempio, se si deposita un milione di lire al tasso di interesse capitalizzato del 5%, il saldo dopo *t* anni sarà uguale a 1000000*EXP(0,05**t*).

Il Programma 9.4 illustra l'uso di EXP per risolvere problemi legati agli interessi.

Controllo della qualità Si supponga di avere una grossa quantità di lampadine e che la vita media di una lampadina sia di *x* ore. La percentuale che una lampadina resti funzionante dopo *t* ore, è di circa EXP(-*t*/*x*). Il Programma 9.5 mostra quante lampadine, su 100 prese come campione, potrebbero restare in vita dopo una serie di ore di attività.

Programma 9.3 *Uso della funzione EXP per il calcolo delle probabilità*

```
REM Calcolo dell'area sotto a una curva normale [9-3]
CLS
INPUT "Media e deviazione standard dei dati"; mean, sd
INPUT "Intervallo su cui calcolare le probabilità"; xLow, xHigh
pi = 4 * ATN(1)
sum = 0
FOR i = xLow TO xHigh STEP sd / 100
  sum = sum + NormalCurve(i)
NEXT i
PRINT "La probabilità che un evento si verifichi nell'intervallo"
PRINT USING "specificato è del ###.## percento"; sum * sd;
PRINT USING " o di 1 su #####"; 100 / (sum * sd)
END

FUNCTION NormalCurve (x)
  SHARED mean, sd, pi
  NormalCurve = (1 / (sd * SQR(2 * pi))) * EXP(-.5 * ((x - mean)_
                / sd) ^ 2)
END FUNCTION

[Esecuzione]
Media e deviazione standard dei dati? 80, 10
Intervallo su cui calcolare le probabilità? 95, 100
La probabilità che un evento si verifichi nell'intervallo
specificato è del 4.50 percento o di 1 su 22
```

Programma 9.4 *Uso della funzione EXP per il calcolo di un interesse capitalizzato*

```
REM Calcolo di un interesse capitalizzato  [9-4]
CLS
INPUT "Somma depositata"; amount
INPUT "Tasso annuo dell'interesse capitalizzato"; intr
INPUT "Quantità di anni trascorsi"; years
IF intr > 1 THEN intr = intr / 100
balance = amount * EXP(intr * years)
PRINT USING "Il saldo finale è $$##,###.##"; balance
END

[Esecuzione]
Somma depositata? 2000
Tasso annuo dell'interesse capitalizzato? 8
Quantità di anni trascorsi? 20
Il saldo finale è   $9,906.06
```

Il risultato potrebbe essere sorprendente, dato che dopo il periodo di vita media solo il 37% delle lampadine risulta funzionante.

FUNZIONI LOGARITMICHE

Esiste una funzione logaritmica corrispondente a ciascuna funzione esponenziale. Per qualsiasi base *b*,

$LOG_b(x)$

è la potenza a cui deve essere elevato *b* per ottenere *x*. QBasic dispone della funzione logaritmica LOG per la base e, conosciuta anche come *logaritmo naturale*. Ad esempio, LOG(2) restituisce 0,69314718, LOG(e) restituisce 1 e LOG(1) restituisce 0.

Programma 9.5 *Uso della funzione EXP per controllare la qualità*

```
REM Quante lampadine rimangono?  [9-5]
DEF FNLeft (t, avgL) = 100 * EXP(-t / avgL)
CLS
INPUT "Vita media (in ore) di una lampadina"; avgLife
PRINT "Usando 100 lampadine come campione, si avranno"
FOR i = avgLife / 2 TO 3 * avgLife STEP avgLife / 2
  PRINT USING "## rimaste dopo ##### ore"; FNLeft(i, avgLife); i
NEXT i
END

[Esecuzione]
Vita media (in ore) di una lampadina? 5000
Usando 100 lampadine come campione, si avranno
61 rimaste dopo  2500 ore
37 rimaste dopo  5000 ore
22 rimaste dopo  7500 ore
14 rimaste dopo 10000 ore
 8 rimaste dopo 12500 ore
 5 rimaste dopo 15000 ore
```

Il logaritmo in base *b* può essere ottenuto da LOG usando la formula

$LOG_b(x) = LOG(x) / LOG(b)$

In particolare

$LOG_{10}(x) = LOG(x) / LOG(10)$
$= LOG(x) / 2.302585$
$= .4342945 * LOG(x)$

In precisione doppia, $LOG_{10}(x) = .4342944819032518 * LOG(x)$

Interesse capitalizzato Un investimento al tasso di interesse *r* capitalizzato viene incrementato *n* volte in LOG(*n*)/*r* anni. Ad esempio, un investimento a un tasso di interesse capitalizzato pari all'8%, si triplica in LOG(3)/.08 anni (cioè 13,7326536084 anni).

TECNICHE PER LA RAPPRESENTAZIONE GRAFICA DELLE FUNZIONI

Nessun programma può garantire la possibilità di generare un buon grafico per qualsiasi funzione. Indipendentemente da come sia stato scritto il programma, un matematico può escogitare una funzione che non può essere rappresentata correttamente. Il programma presentato in questa sezione dispone di strumenti che consentono di fronteggiare alcune delle idiosincrasie comunemente incontrate quando si rappresentano graficamente delle funzioni, e genera un grafico adeguato per molti tipi di funzione.

Il Programma 9.6 è un programma rudimentale per la rappresentazione grafica delle funzioni. Si deve specificare il *dominio* (l'intervallo sull'asse x sopra cui il programma deve tracciare la funzione) e la *scala* (il valore massimo e minimo sull'asse y). Il programma impartisce il comando WINDOW appropriato, traccia le coordinate degli assi e traccia quindi 640 punti sul grafico.

Il Programma 9.6 ha le seguenti limitazioni:

1. il programma deve essere modificato ogni volta che si considera una nuova funzione. Teoricamente, si dovrebbe fornire all'utente la possibilità di specificare interattivamente la funzione desiderata durante l'esecuzione del programma;
2. l'asse x, l'asse y o entrambi potrebbero non apparire sullo schermo. Ciò si verifica, ad esempio, quando *xBasso* o *yBasso* è un numero positivo;
3. si potrebbe non essere in grado di fornire dei valori corretti per *xBasso* e *yBasso*. Nei casi più estremi, una scelta sbagliata genera uno schermo vuoto;
4. ci potrebbero essere alcuni punti nel dominio in cui la funzione è indefinita (come *x*=0 per la funzione 1/*x*), o il cui valore è troppo alto e causa quindi un errore di overflow (ad esempio, *x*=2000 nella funzione EXP(*x*)). Entrambe le situazioni interrompono il programma e potrebbero bloccare il sistema;
5. se i valori nel grafico crescono rapidamente, due punti successivi potrebbero apparire molto distanti tra loro.

Programma 9.6 *Un programma grafico rudimentale*

```
REM Rappresentazione grafica di una funzione  [9-6]
DEF FNF (x) = 1 / x
'Richiede il dominio di una funzione
CLS
INPUT "Il grafico deve iniziare con x = ", startGraph
INPUT "Il grafico deve finire con x = ", endGraph
'Richiede l'intervallo dei valori da rappresentare
INPUT "Qual è il limite inferiore"; yMin
INPUT "Qual è il limite superiore"; yMax
'Inizializza lo schermo
SCREEN 2
VIEW (0, 0)-(639, 180)  'Lascia lo spazio per un messaggio
WINDOW (startGraph, yMin)-(endGraph, yMax)
LINE (startGraph, 0)-(endGraph, 0)     'Traccia l'asse x
LINE (0, yMin)-(0, yMax)               'Traccia l'asse y
'Traccia il grafico
increment = (endGraph - startGraph) / 639
FOR x = startGraph TO endGraph STEP increment
  y = FNF(x)
  PSET (x, y)
NEXT x
END
```

Il Programma 9.7 è un programma sofisticato per la rappresentazione grafica delle funzioni e corregge le cinque limitazioni del Programma 9.6 nei modi seguenti:

1. il Programma 9.7, che deve essere salvato su disco con il nome GRAPH.BAS, scrive in realtà un altro programma, denominato EVALUATE.BAS, che viene utilizzato per calcolare tutti i valori della funzione. Dopo aver specificato la funzione, GRAPH scrive il secondo programma EVALUATE che calcola la funzione nell'intervallo di valori forniti dall'utente. Il primo programma, GRAPH, riempie un array con i valori su cui deve essere calcolata la funzione, ed esegue quindi EVALUATE passando l'array dei valori x. EVALUATE riempie un secondo array con tutti i valori della funzione. L'ultimo comando impartito da EVALUATE esegue nuovamente GRAPH.BAS passandogli l'array contenente i valori della funzione. Si noti, nel programma GRAPH, che la variabile *flag%* viene usata per eseguire una sola volta gli enunciati associati alla condizione IF. Quando EVALUATE esegue GRAPH, *flag%* indica a GRAPH che gli array sono stati riempiti e che la funzione può essere tracciata. Il comando CHAIN consente di richiamare un programma da un altro, e la parola chiave COMMON consente di condividere gli array e la variabile *flag%*;

2. per garantire che l'asse y appaia sullo schermo, l'enunciato WINDOW nel Programma 9.7 usa i valori x inseriti dall'utente per impostare i limiti orizzontali solo quando questi sono, rispettivamente, negativi e positivi. In tutti gli altri casi, uno dei limiti viene sostituito da un numero che garantisce che i limiti

orizzontali abbiano segni differenti. Per esempio, se il limite inferiore dell'asse x è un valore positivo, questo limite viene sostituito con un numero che assegni dei valori negativi almeno al 5% della porzione visibile dell'asse x (ciò assicura che l'asse y appaia sullo schermo). Un calcolo e una sostituzione analoga viene effettuata per i valori dell'asse y nel caso in cui non vengano inseriti direttamente dall'utente;

Programma 9.7 *Un programma sofisticato per la rappresentazione grafica delle funzioni*

```
REM Traccia una funzione specificata dall'utente [9-7]
REM Ci si assicuri di assegnare a questo programma il
REM nome GRAPH.BAS
COMMON flag%, x(), y()          'Variabili condivise
CONST Mode = 2                  'Modalità video
CONST MaxX = 639                'Limite di destra
CONST MaxY = 199                'Limite inferiore
CONST Overflow = 3.402822E+38   'Flag di overflow
CONST True = -1
CONST False = 0
IF flag% = 0 THEN
    CLS
    DIM x(0 TO MaxX + 1)
    DIM y(0 TO MaxX + 1)
    INPUT "Funzione: y = ", function$
    INPUT "Il grafico deve iniziare con x = ", x(0)
    INPUT "Il grafico dovrebbe finire con x = ", x(MaxX)
    increment = (x(MaxX) - x(0)) / (MaxX + 1)
    FOR i% = 1 TO MaxX - 1          'Riempie l'array con i valori x
      x(i%) = x(i% - 1) + increment
    NEXT i%
'Scrive un programma per riempire l'array con i valori y
    OPEN "EVALUATE.BAS" FOR OUTPUT AS #1
    PRINT #1, "COMMON flag%, x(), y()"
    PRINT #1, "ON ERROR GOTO Errhandler"
    PRINT #1, "FOR i% = 0 TO " + STR$(MaxX)
    PRINT #1, "  x = x(i%)"
    PRINT #1, "  y(i%) = " + function$
    PRINT #1, "NEXT i%"
    PRINT #1, "CHAIN " + CHR$(34) + "GRAPH" + CHR$(34)
    PRINT #1, "Errhandler:"
    PRINT #1, "IF ERR THEN"
    PRINT #1, "    y(i%) = " + STR$(Overflow)
    PRINT #1, "    RESUME NEXT"
    PRINT #1, "END IF"
    CLOSE #1
    flag% = -1
    PRINT : PRINT "Elaborazione in corso. Attendere..."
    CHAIN "EVALUATE"             'Richiama il nuovo programma
```

```
  ELSE
    ON ERROR GOTO SkipPoint      'Se non si può tracciare un punto
    CALL SetAxes(x(), y(), yLow, yHigh)
    i% = 0
    pointOk% = False
    DO WHILE i% <= MaxX
      DO WHILE NOT pointOk% AND i% <= MaxX
        IF y(i%) <> Overflow AND y(i%) >= yLow AND y(i%) <= yHigh THEN
            PSET (x(i%), y(i%))
            pointOk% = True
        END IF
        i% = i% + 1
      LOOP
      DO WHILE pointOk% AND i% <= MaxX
        IF y(i%) <> Overflow AND y(i%) >= yLow AND y(i%) <= yHigh THEN
            LINE -(x(i%), y(i%))
            pointOk% = False
        END IF
        i% = i% + 1
      LOOP
    LOOP
END IF
END

SkipPoint:
pointOk% = False
RESUME NEXT

SUB ScaleExtrema (lo, hi)
  REM Forza i valori massimi e minimi
  IF lo >= 0 THEN lo = -hi / 20
  IF hi <= 0 THEN hi = -lo / 20
END SUB

SUB SetAxes (x(), y(), yLow, yHigh)
  REM Imposta la scala e traccia gli assi
  xLow = x(0)
  xHigh = x(MaxX)
  CALL ScaleExtrema(xLow, xHigh)
  PRINT : PRINT "Vuoi specificare i limiti per il ";
  PRINT "grafico (S/N)"
  PRINT "  (Se no, vengono calcolati e usati i valori"
  PRINT "   massimo e minimo della funzione.)"
  answer$ = UCASE$(INPUT$(1))
  IF answer$ = "S" THEN
      INPUT "Qual è il limite inferiore"; yLow
      INPUT "Qual è il limite superiore"; yHigh
    ELSE
      yLow = y(0)
      yHigh = y(0)
```

```
        FOR i% = 1 TO MaxX
          SELECT CASE y(i%)
            CASE Overflow
            CASE IS < yLow: yLow = y(i%)
            CASE IS > yHigh: yHigh = y(i%)
            CASE ELSE
          END SELECT
        NEXT i%
        CALL ScaleExtrema(yLow, yHigh)
    END IF
    SCREEN Mode
    VIEW (0, 0)-(MaxX, MaxY * .9)
    WINDOW (xLow, yLow)-(xHigh, yHigh)
    LINE (xLow, 0)-(xHigh, 0)     'Draw x-axis
    LINE (0, yLow)-(0, yHigh)     'Draw y-axis
END SUB

[Esecuzione]
Funzione: y = (x/300)*(x^2-45)*(x^2-10)
Il grafico deve iniziare con x = -7
Il grafico dovrebbe finire con x = 7

Elaborazione in corso. Attendere...

Vuoi specificare i limiti per il grafico (S/N)?
  (Se no, vengono calcolati e usati i valori"
   massimo e minimo della funzione.)"

[L'utente ha premuto N]
(Il grafico è riportato in Figura 9.3)
```

3. il programma fornisce all'utente la possibilità di decidere se specificare i limiti superiore e inferiore o se lasciare questa incombenza al programma. Il programma imposta i limiti esaminando tutti i valori della funzione, prelevando il valore più alto e quello più basso, e utilizzandoli per impostare le coordinate naturali per il grafico;

4. EVALUATE e GRAPH utilizzano una routine di rilevamento degli errori per gestire i valori indefiniti. L'enunciato ON ERROR GOTO ErrHandler indica al programma di trasferire il controllo alla subroutine ErrHandler ogni volta che si verifica una errore. Quando una funzione non può essere calcolata per certi valori di *x*, o quando a LINE e PSET viene passato un numero molto alto, si verifica un errore. Se la funzione non può essere calcolata, viene assegnato il valore costante Overflow all'elemento dell'array corrispondente; il ciclo FOR continua quindi a calcolare la funzione per i valori x rimanenti. Successivamente, durante la rappresentazione grafica della funzione, il programma salta qualsiasi elemento dell'array contrassegnato con la costante Overflow;

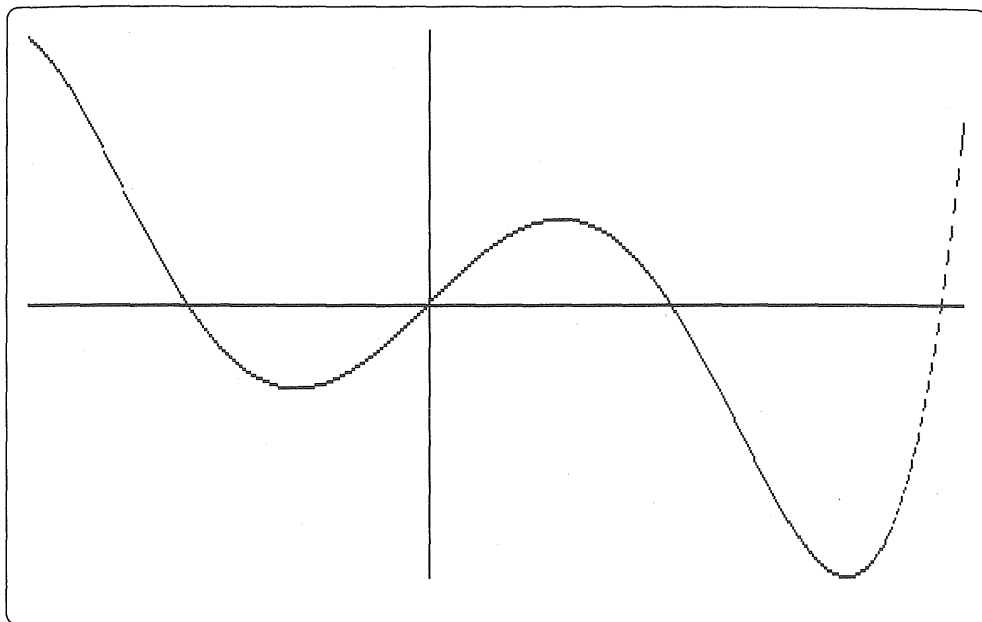



Figura 9.3 *Rappresentazione grafica della funzione f(x)=(x/300)(x^2-45)(x^2-10) generata dal Programma 9.7*

5. per evitare spazi vuoti estesi, il Programma 9.7 non traccia solo i punti, ma usa il comando LINE -(*x,y*) per collegarli con una linea. Questo enunciato traccia una linea che si estende dall'ultimo punto indirizzato al punto (*x,y*). Tuttavia, ci sono alcune eccezioni a questa modalità operativa. Il primo punto tracciato deve essere visualizzato con PSET, dato che l'ultimo punto indirizzato non è un valore della funzione. Quindi, LINE viene usato per collegare punti validi. Se il programma non può tracciare un determinato punto nella finestra corrente, lo salta e utilizza PSET per visualizzare il punto successivo. Viene quindi utilizzato nuovamente il comando LINE fino a quando non viene incontrato un altro punto non valido.

NUMERI CASUALI

Si consideri una serie specifica di numeri. Si può dire che una procedura seleziona un numero a caso da questa serie, se tutti i numeri della serie hanno le stesse probabilità di essere selezionati e se non si può prevedere questo numero in anticipo. Alcuni esempi sono riportati nella Tabella 9.1.

La funzione RND, che opera come la freccia in Figura 9.4, fornisce a QBasic la capacità di selezionare un numero a caso da una serie di numeri. Il valore restituito da questa funzione è un numero compreso tra 0 e 1 (escluso). Ogni volta che RND appare in un programma, genera un numero differente, e qualsiasi numero maggiore o uguale a zero e minore di 1 ha le stesse probabilità di essere generato.

A questo punto, ci sono alcune considerazioni positive e negative da tener presente. Di positivo c'è il fatto che i valori generati dalla funzione RND sono esattamente ciò che occorre per sviluppare applicazioni che prevedano la presenza di eventi casuali. Di negativo, invece, c'è il fatto che le ripetizioni successive di RND generano sempre la stessa sequenza di numeri (questa caratteristica di RND è intenzionale ed è importante durante il collaudo di programmi che svolgono simulazioni). Tuttavia, QBasic dispone di un altro comando, RANDOMIZE TIMER, che utilizza l'orologio interno del computer per cambiare la sequenza dei numeri generati da RND. Questa sequenza non è veramente casuale poiché ogni numero determina, in realtà, quello successivo. Tuttavia, la sequenza di numeri generata da RND appare come una sequenza generata casualmente.

Per qualsiasi sottointervallo dell'intervallo (0,1), ad esempio da 1/4 a 1/2, le probabilità di generare un numero in quel sottointervallo sono le stesse per qualsiasi altro sottointervallo della stessa lunghezza (ad esempio, da 1/2 a 3/4). La sequenza dei numeri generati da RND è detta pseudo-casuale.

Tabella 9.1 *Metodi di selezione di numeri casuali da una serie di numeri*

Serie	Procedura
1, 2, 3, 4, 5, 6	Tiro di un dado a sei facce
0 o 1	Lancio di una moneta: 0=croce, 1=testa
-1, 0, 1, ..., 36	La ruota di una roulette (-1 indica 00)
1, 2, ..., N	Scrittura di numeri su foglietti di carta ed estrazione da un cappello
Numeri da 0 a 1	La freccia in Figura 9.4

Il Programma 9.8 usa la funzione RND per selezionare un numero casuale da ciascuna delle serie di numeri riportate nella Tabella 9.1. Dato che il valore di RND è compreso tra 0 e 1 (1 escluso), l'espressione 6*RND genera un numero compreso tra 0 e 6 (6 escluso). Quindi, INT(6*RND) può essere 0, 1, 2, 3, 4 o 5, mentre INT(6*RND)+1 può essere 1, 2, 3, 4, 5 o 6. Il programma simula il lancio di una moneta per determinare un evento che ha il 50% di probabilità di verificarsi. La ruota della roulette viene gestita in modo analogo al tiro dei dadi. In generale, il valore di INT(*n**RND) restituisce un numero che può essere 0, 1, 2,...., *n*-1, mentre il valore generato da INT(*n**RND)+*m*

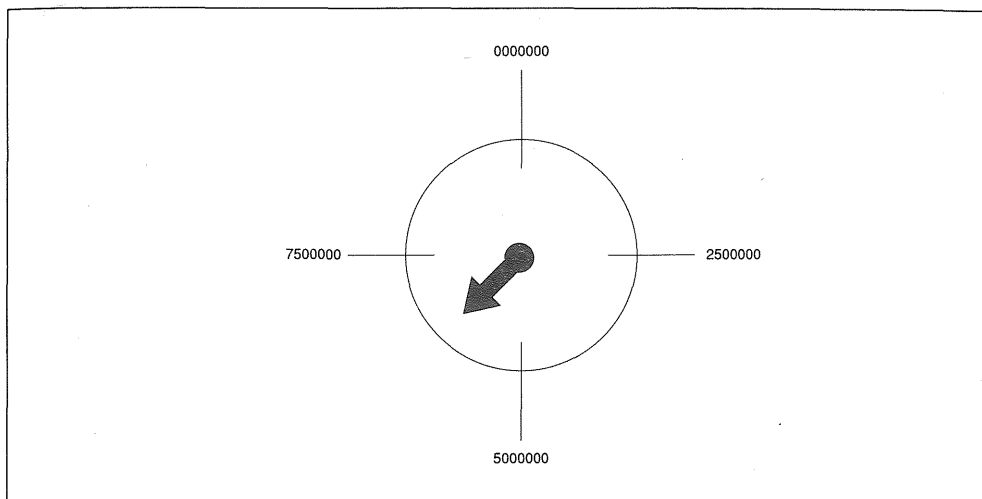



Figura 9.4 *Una freccia per selezionare un numero casuale compreso tra 0 e 1*

fornisce un numero compreso rea m e $m+n-1$. Si può quindi usare questa tecnica per selezionare un intero da una qualsiasi serie di numeri interi consecutivi.

Il Programma 9.9 usa la funzione RND per mescolare un mazzo di carte e per pescarne cinque dalla cima del mazzo. Il programma inserisce inizialmente 52 carte in un nuovo array disponendo prima i cuori, quindi i quadri, i fiori e i picche. Quindi, le carte da 1 a 13 saranno cuori, quelle da 14 a 26 quadri, quelle da 27 a 39 fiori e quelle da 40 a 52 picche. Ogni seme viene disposto in ordine, dall'asso al re. Il programma identifica ogni carta usando una stringa che consiste della denominazione e del seme. I simboli per i cuori, quadri, fiori e picche hanno, rispettivamente, valore ASCII 3, 4, 5 e 6 e possono essere visualizzati con le funzioni CHR$(3), CHR$(4), CHR$(5) e CHR$(6). Il programma mischia le carte scambiando la posizione di ogni carta nell'array con un evento casuale.

Il Programma 9.9 presenta un metodo per selezionare cinque carte da un mazzo di 52 carte. Questo è un caso speciale del problema che deriva dal dover selezionare m oggetti da una serie di n oggetti o; nel caso specifico, di selezionare m interi da una serie di numeri compresi tra 1 e n. Il Programma 9.10 presenta un elegante algoritmo per la risoluzione di questo problema. Questo algoritmo, inoltre, ordina la serie degli m numeri. Per capire pienamente il funzionamento di questo algoritmo, si riproducano su carta i calcoli svolti dal Programma 9.10.

Si è visto finora come utilizzare la funzione RND per delle simulazioni, pescando cinque carte da un mazzo ed estraendo dei numeri che potrebbero essere generati da una roulette. Seguono altri tipi di utilizzo di questa funzione.

Programma 9.8 *Uso di RND*

```
REM Seleziona un numero casuale  [9-8]
RANDOMIZE TIMER
d% = INT(6 * RND) + 1
PRINT "Il numero fornito dal dado è"; d%
IF RND < .5 THEN result$ = "testa" ELSE result$ = "croce"
PRINT "La moneta indica "; result$
r% = INT(38 * RND) - 1
PRINT "La pallina si è fermata sul numero";
IF r% = -1 THEN PRINT " 00" ELSE PRINT r%
PRINT "La freccia indica il numero";
PRINT USING " .###"; RND
END

[Esecuzione] (i risultati variano)
Il numero fornito dal dado è 4
La moneta indica testa
La pallina si è fermata sul numero 32
La freccia indica il numero .072
```

Verifica della correttezza e dell'efficienza di un programma Se si selezionano casualmente delle voci di menu di un programma, si evita qualsiasi propensione da parte del tester.

Analisi numerica Le aree al di sotto di certe curve, come nel caso della curva normale, esprimono delle informazioni. Un metodo per determinare l'area sotto una curva consiste nel racchiudere l'area in un rettangolo, selezionare casualmente dei punti dal rettangolo, e calcolare la percentuale di punti che cade sotto la curva. Si può quindi determinare l'area sottostante alla curva in base alla percentuale di punti così rilevata.

Divertimento Si possono scrivere diversi tipi di giochi come, ad esempio, blackjack. Inoltre, si possono sviluppare dei programmi che simulano giochi di fortuna e analizzano diverse strategie.

Decisioni Nella Teoria dei Giochi, un ramo della matematica a volte applicato all'economia, alcune strategie usano la funzione RND per simulare delle decisioni.

CREAZIONE DI CARATTERI MATEMATICI PERSONALIZZATI

In modalità grafica, si può creare un set di caratteri personalizzato per sostituire i caratteri che si trovano nella metà superiore della tabella ASCII (Da 128 a 255). Dato

che il set di caratteri viene caricato in memoria, si vedrà innanzi tutto come specificare le locazioni di memoria necessarie.

Programma 9.9 *Uso di RND per cambiare casualmente la posizione di una serie di numeri*

```
REM Mischia un mazzo di carte  [9-9]
DIM card$(1 TO 52)           'Array per il mazzo di carte
CLS
CALL SetUpDeck(card$())      'Nuovo mazzo di carte
CALL Shuffle(card$())        'Mischia il mazzo
CALL DisplayFive(card$())    'Mostra le prime 5 carte
REM ----------- Dati per le carte del mazzo
DATA A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K
END

SUB DisplayFive (card$())
  FOR i = 1 TO 5
    PRINT card$(i) + "  ";
  NEXT i
  PRINT
END SUB

SUB SetUpDeck (card$())
  hearts = 3: spades = 6
  ace = 1: king = 13
  FOR suit% = hearts TO spades
    RESTORE
    FOR denomination% = ace TO king
      READ denom$
      cardNumber = 13 * (suit% - hearts) + denomination%
      card$(cardNumber) = denom$ + CHR$(suit%)
    NEXT denomination%
  NEXT suit%
END SUB

SUB Shuffle (card$())
  RANDOMIZE TIMER
  FOR i = 1 TO 52
    SWAP card$(i), card$(INT(52 * RND) + 1)
  NEXT i
END SUB

[Esecuzione] (I risultati variano)
8♥  9♠  6♦  K♠  4♣
```

Programma 9.10 *Uso di RND per selezionare casualmente una sottoserie di una serie di numeri*

```
REM Sceglie m numeri tra 1 e n  [9-10]
CLS
INPUT "Numeri compresi tra 1 e "; n
INPUT "Quantità di numeri da scegliere"; m
RANDOMIZE TIMER
needed = m
remaining = n
FOR i% = 1 TO n
  IF RND < needed / remaining THEN
     PRINT i%;
     needed = needed - 1
  END IF
  remaining = remaining - 1
NEXT i%
PRINT
END

[Esecuzione] (i risultati variano)
Numeri compresi tra 1 e ? 10
Quantità di numeri da scegliere? 3
 1  7  8
```

SPECIFICA DELLE LOCAZIONI DI MEMORIA

In linguaggio informatico, la lettera K indica 1024 byte. Quindi, 64K equivale a 65.536 byte e K^2 a 1.048.576 byte. Le locazioni della memoria di base di un computer sono numerate da 0 a K^2-1, cioè da 0 a 1.048.575. Alcuni blocchi di memoria da 64K sono chiamati segmenti:

il segmento 0 consiste delle locazioni di memoria 0, 1, 2, ..., 65.535
il segmento 1 consiste delle locazioni di memoria 16, 17, 18, ..., 65.551
il segmento 3 consiste delle locazioni di memoria 32, 33, 34, ..., 65.657
..
..
il segmento m consiste delle locazioni di memoria 16*m, 16*m+1, 16*m+2, ..., 16*m+65.535

All'interno di ogni segmento, si dice che le locazioni hanno distanza (offset) 0, 1, 2, ..., 65.535. Si può specificare qualsiasi locazione di memoria tramite un numero di segmento e una distanza in quel segmento. Questi due numeri vengono generalmente scritti nella forma *segmento:distanza.* Una determinata locazione di memoria può essere rappresentata con diverse forme di tipo *segmento:distanza.* Ad esempio, la locazione 35 può essere indicata come 0:35, 1:19 o 2:3.

In qualsiasi momento, si può dichiarare un segmento di memoria come segmento corrente. Gli enunciati e le funzioni che leggono e scrivono dei dati in memoria, identificano una locazione tramite la sua distanza dal segmento corrente. L'enunciato

```
DEF SEG = m
```

dichiara il segmento *m* come segmento corrente.

Ogni locazione di memoria contiene un byte, composto da otto bit che possono avere valore uguale a zero a uno. Questo ottetto costituisce la rappresentazione binaria di un numero compreso tra 0 e 255. Quindi, si può affermare che ciascuna locazione di memoria contiene un numero compreso tra 0 e 255. La funzione

```
PEEK(n)
```

restituisce il numero contenuto nella locazione di memoria che si trova a una distanza *n* nel segmento corrente. L'enunciato

```
POKE n, r
```

inserisce il numero *r* nella locazione di memoria che dista *n* dall'inizio del segmento corrente. Il numero *r* può essere specificato sia in notazione decimale che esadecimale. I numeri scritti in esadecimale devono essere preceduti da &H. L'uso della notazione esadecimale semplifica l'inserimento di un byte in una determinata locazione di memoria.

Le locazioni di memoria nel segmento 0 forniscono delle informazioni sui componenti hardware del computer e sullo stato della tastiera, dello schermo e di alcune porzioni speciali di memoria. In particolare, le locazioni di memoria 124, 125, 126 e 127 puntano il segmento e la relativa distanza dell'inizio della porzione di memoria contenete i caratteri del codice ASCII esteso. Questa porzione di memoria è chiamata *tabella di caratteri.* La distanza e il segmento usati per l'inizio della tabella dei caratteri vengono ricavati, rispettivamente, dalle espressioni PEEK(124)+256*PEEK(125) e PEEK(126)+256*PEEK(127).

SPECIFICA DEI CARATTERI IN MODALITÀ GRAFICA

In modalità SCREEN 1 e 2, ciascun carattere viene visualizzato in una matrice rettangolare di 8 pixel per lato. La Figura 9.5 mostra il carattere 1/3 definito dall'utente. A destra di ciascuna riga della matrice è riportata la stringa binaria che definisce la riga stessa. Il valore uno indica un pixel attivo, mentre uno zero indica un pixel non attivo. La sequenza delle otto stringhe binarie definisce il carattere 1/3.

Per definire un carattere del codice ASCII esteso, si proceda nel modo seguente:

1. creare una variabile stringa a lunghezza fissa in cui memorizzare la tabella dei caratteri. Conviene utilizzare una stringa a lunghezza fissa dato che, una volta assegnata la sua locazione di memoria, questa non viene cambiata da QBasic. Inoltre, sono disponibili le funzioni VARSEG e VARPTR che consentono di specificare il segmento e la distanza del valore della variabile;
2. impostare le locazioni di memoria 124, 125, 126 e 127 del segmento 0 in modo che puntino alla variabile stringa a lunghezza fissa. Se la variabili inizia alla locazione *s:f* nel segmento *segmento:distanza*, i valori da inserire nelle locazioni 124, 125, 126 e 127 sono, rispettivamente, *f*MOD 256, *f*\256, *s*MOD 256 e *s*\256;
3. inserire gli otto numeri binari che descrivono il carattere 128 nelle prime otto locazioni della stringa a lunghezza fissa. Inserire gli otto numeri binari che descrivono il carattere 129 nelle otto locazioni successive della tabella dei caratteri. Continuare con questa procedura fino ad aver inserito tutti i 128 caratteri nella tabella.

Il Programma 9.11 definisce il carattere 128 come frazione 1/3 e il carattere 129 come triangolo rettangolo pieno. Benché siano necessari solo 16 byte di memoria per definire questi due caratteri, il programma riserva una tabella di caratteri di 1024 byte in modo che possa eventualmente contenere le descrizioni di tutti i 128 caratteri. Le informazioni relative a questi caratteri sono contenute negli enunciati DATA e vengono inseriti nella tabella dei caratteri tramite il comando POKE. Il programma può essere facilmente modificato per definire caratteri addizionali.

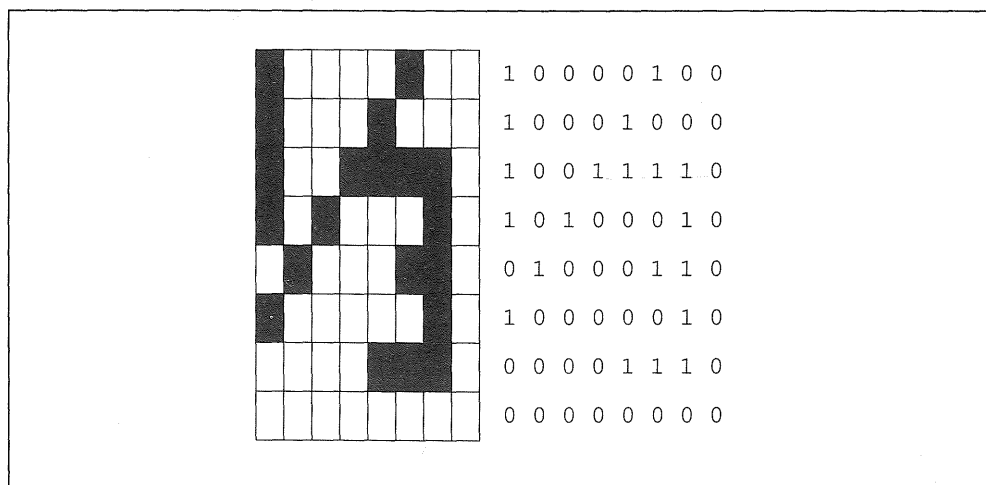



Figura 9.5 *Come appare il carattere 1/3 sullo schermo*

La procedura necessaria per disegnare un carattere è molto simile a quella esaminata per l'impostazione di un motivo di riempimento. Si veda la discussione sul comando PAINT per ulteriori dettagli. La Figura 9.5 mostra la rappresentazione binaria dei numeri usati per definire il carattere 1/3.

Programma 9.11 *Definizione dei caratteri*

```
REM Caratteri definiti dall'utente  [9-11]
SCREEN 1
DEF SEG = 0
DIM store(0 TO 3) AS INTEGER
FOR i = 0 TO 3
  store(i) = PEEK(124 + i)
NEXT i
REM Riempie quattro byte con nuovi valori
CONST numChars = 128
CONST length = 8 * numChars
DIM chars AS STRING * length
offset = VARPTR(chars)
POKE 124, offset MOD 256
POKE 125, offset \ 256
segment = VARSEG(chars)
POKE 126, segment MOD 256
POKE 127, segment \ 256
REM Inserisce in memoria i dati per i caratteri
DEF SEG
FOR i = 0 TO 15
  READ a%
  POKE offset + i, a%
NEXT i
REM Ogni comando DATA descrive un carattere
DATA 132,136,158,162,70,130,14,0: 'un terzo
DATA 2,6,14,30,62,126,254,0: 'triangolo pieno
REM Visualizza i nuovi caratteri definiti
PRINT CHR$(128) + "  ";
PRINT CHR$(129)
REM Ripristina lo stato dei quattro byte di memoria
DEF SEG = 0
FOR i = 0 TO 3
  POKE 124 + i, store(i)
NEXT i
END

[Esecuzione]
1/3 ▲
```

CARATTERI MATEMATICI

Il codice ASCII esteso riportato nell'Appendice A contiene alcuni simboli matematici. Alcuni di questi caratteri sono descritti nella Tabella 9.2.

Tabella 9.2 *Alcuni simboli matematici del codice ASCII esteso*

Valore ASCII	Descrizione	Valore ASCII	Descrizione
171	1/2	239	Intersezione
172	1/4	241	Segno più o meno
224-235	Lettere greche	244-245	Segno integrale
236	Infinito	251	Simbolo di radice quadrata
238	Appartenente all'insieme	253	2 come esponente

SALVATAGGIO DEI GRAFICI

La scheda grafica riserva 4000 locazioni per memorizzare il contenuto dello schermo. Dopo aver tracciato un grafico, si può salvare il contenuto di queste locazioni un un file e ricaricarlo successivamente per riprodurre il grafico. L'enunciato

```
DEF SEG = &HB800
BSAVE nomefile, 0, 4000
```

salva i byte contenuti in queste locazioni nel file specificato. Quando ci si trova nella stessa modalità SCREEN in cui è stato salvato il file, si possono usare i comandi

```
DEF SEG = &HB800
BLOAD nomefile
```

per ripristinare la schermata originale.

MATRICI

Una matrice *m*n* è un array a due dimensioni dichiarato da un enunciato nella forma

```
DIM nomeArray(1 TO m, 1 TO n)
```

La matrice avrà *m* righe e *n* colonne. Alcune versioni del BASIC dispongono di funzioni incorporate per la somma, la moltiplicazione e l'inversione delle matrici.

Benché QBasic non disponga di queste funzioni, consente di definirle facilmente come sottoprogrammi.

ADDIZIONE DI MATRICI

Bisogna dimensionare tutte e tre le matrici passate al Sottoprogramma 9.12 (anche quella che dovrà contenere la matrice risultante) prima di chiamare il sottoprogramma. All'interno del sottoprogramma, le due matrici da sommare sono denominate *first* e *second*, e la matrice risultante è chiamata *sum*.

Il sottoprogramma inizia verificando che le tre matrici passate al sottoprogramma abbiano lo stesso numero di righe e di colonne. Dopo questa verifica, il sottoprogramma somma le matrici usando una coppia di cicli FOR...NEXT nidificati.

Sottoprogramma 9.12 *Somma di due matrici*

```
SUB MatrixAddition (first(), second(), sum())       '[9-12]
  'si presume che ogni matrice sia stata dimensionata
  'nella forma nomeArray(1 TO m, 1 TO n)
  rowsFirst = UBOUND(first, 1)
  colsFirst = UBOUND(first, 2)
  rowsSecond = UBOUND(second, 1)
  colsSecond = UBOUND(second, 2)
  rowsSum = UBOUND(sum, 1)
  colsSum = UBOUND(sum, 2)
  'Verifica che le due matrici abbiano la stessa dimensione
  IF (rowsFirst <> rowsSecond) OR (colsFirst <> colsSecond) THEN
      PRINT "dimensioni non appropriate per l'addizione"
      EXIT SUB
  END IF
  'Verifica che la matrice risultante abbia una dimensione
  'appropriata
  IF (rowsFirst <> rowsSum) OR (colsFirst <> colsSum) THEN
      PRINT "La matrice risultante non ha una dimensione appropriata."
      EXIT SUB
  END IF
  'Esegue l'addizione
  FOR i1% = 1 TO rowsSum
    FOR i2% = 1 TO colsSum
      sum(i1%, i2%) = first(i1%, i2%) + second(i1%, i2%)
    NEXT i2%
  NEXT i1%
END SUB
```

Sottoprogramma 9.13 *Moltiplicazione di due matrici*

```
SUB MatrixMultiply (first(), second(), product())     '[9-13]
  'si presume che ogni matrice sia stata dimensionata
  'nella forma nomeArray(1 TO m, 1 TO n)
  rowsFirst = UBOUND(first, 1)
  colsFirst = UBOUND(first, 2)
  rowsSecond = UBOUND(second, 1)
  colsSecond = UBOUND(second, 2)
  rowsProduct = UBOUND(product, 1)
  colsProduct = UBOUND(product, 2)
  'Verifica che le due matrici possano essere moltiplicate
  IF colsFirst <> rowsSecond THEN
      PRINT "Dimensioni non appropriate per la moltiplicazione"
      EXIT SUB
  END IF
  'Verifica che il prodotto abbia una dimensione appropriata
  IF (rowsFirst <> rowsProduct) OR (colsSecond <> colsProduct) THEN
      PRINT "Dimensioni non appropriate per il prodotto"
      EXIT SUB
  END IF
  'Esegue la moltiplicazione
  FOR i1% = 1 TO rowsProduct
    FOR i2% = 1 TO colsProduct
      total = 0
      FOR i3% = 1 TO colsFirst
        total = total + first(i1%, i3%) * second(i3%, i2%)
      NEXT i3%
      product(i1%, i2%) = total
    NEXT i2%
  NEXT i1%
END SUB
```

MOLTIPLICAZIONE DI MATRICI

Bisogna dimensionare tutte e tre le matrici passate al Sottoprogramma 9.13 (anche quella che dovrà contenere il prodotto) prima di chiamare il sottoprogramma. All'interno del sottoprogramma, le due matrici da moltiplicare sono denominate *first* e *second*, e la matrice risultante è chiamata *product*. Il sottoprogramma inizia verificando che le tre matrici passate abbiano delle dimensioni appropriate; ciò significa che il numero di colonne della prima matrice deve essere uguale al numero di righe della seconda, il numero di colonne della matrice risultante deve essere uguale al numero di colonne della seconda matrice, e il numero di righe della matrice risultante deve essere uguale al numero di righe della prima matrice. Dopo questa verifica, i valori della matrice risultante vengono ottenuti uno alla volta. Ogni valore corrisponde alla somma dei prodotti dei numeri prelevati dala riga appropriata della prima matrice e dalla colonna appropriata della seconda matrice.

INVERSIONE DI MATRICI

Bisogna dimensionare le due le matrici passate al Sottoprogramma 9.14 (anche quella che dovrà contenere la matrice inversa) prima di chiamare il sottoprogramma. All'interno del sottoprogramma, la matrice da invertire è denominata *original#* e la matrice inversa *inverse#*. Il calcolo utilizzato per ottenere l'inverso di una matrice è molto sensibile agli errori di arrotondamento. Per questo motivo è necessario utilizzare delle variabili a precisione doppia.

Il sottoprogramma inizia verificando che le due matrici passate abbiano delle dimensioni appropriate. Entrambe le matrici devono essere quadrate. Il sottoprogramma esegue l'inversione utilizzando una variante dell'algoritmo di eliminazione di Gauss-Jordan. La matrice da invertire viene inserita nella metà di sinistra di una matrice estesa denominata *b#*, la cui metà destra contiene la matrice identica. Il sottoprogramma svolge delle elementari operazioni di riga su *b#* per trasformare la metà di sinistra in una matrice diagonale. Per ridurre al minimo gli errori di arrotondamento, l'algoritmo effettua le divisioni alla fine. Dopo le divisioni appropriate, la metà di destra di *b#* contiene l'inverso della matrice originale.

In questo capitolo sono state esaminate le funzioni necessarie per risolvere dei problemi di tipo scientifico e matematico. Il prossimo e ultimo capitolo di questo libro propone delle funzioni e degli esempi di programmi gestionali.

Sottoprogramma 9.14 *Calcolo della matrice inversa*

```
SUB MatrixInversion (original#(), inverse#())     '[9-14]
  'Si presume che le matrici siano quadrate e che gli
  'indici siano compresi tra 1 e UBOUND(original#())
  'Verifica che le matrici abbiano la stessa dimensione
  IF UBOUND(original#, 1) <> UBOUND(original#, 2) THEN
      PRINT "La matrice da invertire non è quadrata."
      EXIT SUB
  END IF
  IF UBOUND(inverse#, 1) <> UBOUND(inverse#, 2) THEN
      PRINT "La matrice risultante non è quadrata."
      EXIT SUB
  END IF
  IF UBOUND(original#, 1) <> UBOUND(inverse#, 1) THEN
      PRINT "La matrice risultante non ha una dimensione appropriata."
      EXIT SUB
  END IF
  'Dimensiona un array con larghezza doppia di quella data,
  'copia l'array dato nella parte sinistra del nuovo array,
  'e inserisce nella parte destra la matrice identica
  size% = UBOUND(original#, 1)
  DIM b#(size%, 2 * size%)
```

```
  FOR row% = 1 TO size%
    FOR col% = 1 TO size%
      b#(row%, col%) = original#(row%, col%)
    NEXT col%
    b#(row%, size% + row%) = 1
  NEXT row%
  'Usa il metodo di Gauss-Jordan modificato per l'inversione
  FOR i% = 1 TO size%
    'Se l'elemento diagonale è zero, scambia la riga con una
    'successiva per ottenere un elemento diagonale non-zero.
    IF b#(i%, i%) = 0 THEN
      switched% = 0
      FOR testRow% = i% + 1 TO size%
        IF b#(testRow%, i%) <> 0 THEN
          FOR col% = 1 TO 2 * size%
            temp# = b#(i%, col%)
            b#(i%, col%) = b#(testRow%, col%)
            b#(testRow%, col%) = temp#
          NEXT col%
          switched% = -1
        END IF
      NEXT testRow%
      IF NOT switched% THEN
        PRINT "La matrice non ha inverso."
        EXIT SUB
      END IF
    END IF
    'Riduce tutti gli elementi non diagonali nella colonna
    'a zero
    FOR row% = 1 TO size%
      IF row% <> i% THEN
          hold# = b#(row%, i%)
          FOR col% = 1 TO 2 * size%
            b#(row%, col%) = b#(i%, i%) * b#(row%, col%)
            b#(row%, col%) = b#(row%, col%) - hold# * b#(i%, col%)
          NEXT col%
      END IF
    NEXT row%
  NEXT i%
  FOR row% = 1 TO size%
    FOR col% = 1 TO size%
      inverse#(row%, col%) = b#(row%, col% + size%) / b#(row%, row%)
    NEXT col%
  NEXT row%
END SUB
```

CAPITOLO 10

PROGRAMMI GESTIONALI

Nel capitolo precedente sono state presentate delle funzioni per svolgere calcoli matematici. In quest'ultimo capitolo si metteranno in pratica tutte le nozioni apprese nel corso del libro per sviluppare alcuni programmi finanziari e un programma completo per la gestione di un database.

Verranno innanzi tutto esposti alcuni piccoli trucchi di programmazione che consentono di rendere i calcoli più accurati e precisi.

CALCOLI FINANZIARI

Dopo aver motivato l'utilità degli interi lunghi nei calcoli finanziari, questa sezione propone un'analisi dettagliata dell'ammortamento di un prestito.

L'IMPORTANZA DEGLI INTERI LUNGHI

Molti calcoli finanziari risultano più accurati se si considerano le unità elementari della valuta corrente utilizzando le variabili intere lunghe. Questa tecnica non solo si traduce in un'elaborazione più efficace, ma garantisce al tempo stesso un elevato

grado di precisione. Un esempio di questa affermazione è fornito dal Programma 10.1, che utilizza le variabili intere lunghe per calcolare l'interesse ottenuto e per arrotondarlo all'unità più vicina. Nel nostro esempio, si farà riferimento alla relazione fra dollari e centesimi.

Programma 10.1 *Uso delle variabili intere lunghe nei calcoli finanziari*

```
REM Calcolo dell'interesse con gli interi lunghi  [10-1]
CLS
p& = 123456
interestRate = .05
interestEarned& = interestRate * p&
PRINT "L'interesse ottenuto è"; interestEarned& / 100
END

[Esecuzione]
L'interesse ottenuto è 61.73
```

Il Programma 10.2, invece, usa le variabili a precisione doppia per eseguire lo stesso calcolo del Programma 10.1. La funzione FNRound viene usata per arrotondare i numeri a due posizioni decimali. Nonostante FNRound operi nel modo migliore, il suo sforzo è parzialmente vanificato dal modo peculiare in cui QBasic memorizza e visualizza i numeri a precisione doppia.

Nel Programma 10.2 si può visualizzare correttamente l'ammontare dell'interesse tramite un enunciato PRINT USING. Tuttavia, questa è solo una soluzione temporanea; il valore della variabile, infatti, non è preciso e non si otterrebbe il risultato aspettato se si eseguissero ulteriori calcoli con l'interesse ottenuto.

Le variabili non intere, quindi, generano dei problemi con le frazioni di un centesimo. In alcuni casi, soprattutto quando si lavora con numeri estesi, questi errori continuano ad accumularsi fino a raggiungere cifre di una certa entità.

Programma 10.2 *Problemi di arrotondamento con le variabili a precisione doppia*

```
REM Calcolo dell'interesse con numeri a precisione doppia [10-2]
DEF FNRound (x#) = INT(100 * x# + .5) / 100
p# = 1234.56
interestRate = .05
interestEarned# = FNRound(interestRate * p#)
PRINT "L'interesse ottenuto è"; interestEarned#
END

[Esecuzione]
L'interesse ottenuto è 61.72999954223633
```

AMMORTAMENTO DI UN PRESTITO

Il metodo standard per l'ammortamento di un prestito prevede pagamenti mensili costanti fino al suo completo esaurimento. In un ammortamento vengono usati i termini seguenti:

Importo	L'ammontare del prestito
Tasso	Il tasso di interesse fornito in percentuale; ad esempio, 1" o 10,5
Durata	Il numeri di mesi necessari per ammortizzare il debito
Rata	Il pagamento mensile

Dati i valori di tre di queste variabili, si può determinare il valore della quarta tramite una formula matematica. Nella maggior parte dei casi si conosce l'importo, il tasso e la durata, e si deve determinare la rata mensile.

In qualsiasi momento, il *saldo* corrisponde all'importo corrente del prestito, cioè alla quantità di denaro che deve ancora essere pagata per estinguere il debito. Il saldo iniziale è uguale al prestito ottenuto e decrementa fino a zero mano a mano che vengono effettuati i pagamenti mensili. Un pagamento mensile comprende già l'interesse dovuto.

interesse pagato mensilmente	=	tasso di interesse mensile*bilancio all'inizio del mese
riduzione mensile del debito	=	pagamento-interesse pagato mensilmente
saldo alla fine del mese	=	saldo all'inizio del mese-riduzione mensile del debito

Il Programma 10.3 richiede all'utente i termini di un prestito (l'importo, il tasso di interesse annuo, la durata e la rata mensile). Se l'utente omette l'importo, la durata o la rata mensile, il programma lo calcola automaticamente (il calcolo del tasso di interesse sulla base delle altre tre variabili è molto complesso e non è stato implementato). Una volta conosciute tutte le variabili, l'utente può richiedere un piano di ammortamento sulla base dei dati specificati.

Programma 10.3 *Ammortamento di un prestito (output riportato nelle Figure 10.1 e 10.2)*

```
REM Calcola i termini e mostra un piano di ammortamento [10-3]
CLS
DEFDBL A-Z
PRINT "Si digiti 0 per l'importo, la durata o la rata"
PRINT "per calcolare il valore relativo."
INPUT "Importo del prestito"; amount
INPUT "Tasso di interesse annuo (percentuale)"; rate
PRINT "Periodo richiesto (in mesi) per estinguere il debito";
INPUT duration%
```

```
INPUT "Importo di ogni rata"; payment
i = rate / 1200   'Tasso di interesse mensile
IF amount = 0 THEN
    CALL FindAmount
  ELSEIF duration% = 0 THEN
    CALL FindDuration
    CALL FindPayment  'verifica l'esattezza della rata
  ELSEIF payment = 0 THEN
    CALL FindPayment
END IF
PRINT "Vuoi vedere il piano di ammortamento (S/N)?"
answer$ = UCASE$(INPUT$(1))
IF answer$ = "S" THEN CALL Amortize
END

SUB Amortize
  SHARED amount, i, duration%, payment
  CLS
  PRINT "RATA N.    IMPORTO    INTERESSE    RID. DEBITO          SALDO"
  balance = 100 * amount
  payment& = 100 * payment
  FOR payNum% = 1 TO duration%
    interest& = i * balance
    'Se è l'ultima rata, la ricalcola
    IF payNum% = duration% THEN payment& = interest& + balance
    reductionOfLoan& = payment& - interest&
    balance = balance - reductionOfLoan&
    PRINT USING "####    #####,###    "; payNum%; payment& / 100;
    PRINT USING "#####,###      "; interest& / 100;
    PRINT USING "#####,###    "; reductionOfLoan& / 100;
    PRINT USING "#######,###"; balance / 100
    IF (payNum% MOD 12 = 0) AND (payNum% <> duration%) THEN
      PRINT
      PRINT "Premere un tasto per continuare."
      a$ = INPUT$(1)
      CLS
    PRINT "RATA N.    IMPORTO    INTERESSE    RID. DEBITO          SALDO"
    END IF
  NEXT payNum%
END SUB

FUNCTION Ceil (x)
  REM Il numero intero più piccolo maggiore o uguale a x
  Ceil = -INT(-x)
END FUNCTION

SUB FindAmount
  SHARED amount, i, duration%, payment
  amount = (payment / i) * (1 - (1 + i) ^ -duration%)
  PRINT USING "L'ammontare del prestito è #######,###"; amount
END SUB
```

```
SUB FindDuration
  SHARED amount, i, duration%, payment
  duration% = Ceil(LOG(payment / (payment - i * amount)) / LOG(1 + i))
  PRINT USING "Il debito verrà estinto in ### mesi"; duration%
END SUB

SUB FindPayment
  SHARED amount, i, duration%, payment
  payment = (i * amount) / (1 - (1 + i) ^ -duration%)
  PRINT USING "Ogni rata sarà di lire ###,###"; payment
END SUB


[Esecuzione]
Si digiti 0 per l'importo, la durata o la rata
per calcolare il valore relativo.
Importo del prestito? 100000000
Tasso di interesse annuo (percentuale)? 9.5
Periodo richiesto (in mesi) per estinguere il debito? 360
Importo di ogni rata? 0
Ogni rata sarà di lire 840,854
Vuoi vedere il piano di ammortamento (S/N)? S
```

RATA N.	IMPORTO	INTERESSE	RID. DEBITO	SALDO
1	840,854	791,667	49,188	99,950,812
2	840,854	791,277	49,577	99,901,236
3	840,854	790,885	49,969	99,851,266
4	840,854	790,489	50,365	99,800,901
5	840,854	790,090	50,764	99,750,137
6	840,854	789,689	51,166	99,698,972
7	840,854	789,284	51,571	99,647,401
8	840,854	788,875	51,979	99,595,422
9	840,854	788,464	52,390	99,543,032
10	840,854	788,049	52,805	99,490,226
11	840,854	787,631	53,223	99,437,003
12	840,854	787,210	53,645	99,383,359

Premere un tasto per continuare.

Figura 10.1 *Ammortamento dopo un anno*

```
RATA N.     IMPORTO     INTERESSE      RID. DEBITO          SALDO
 349        840,854       75,918         764,936        8,824,709
 350        840,854       69,862         770,992        8,053,717
 351        840,854       63,759         777,096        7,276,621
 352        840,854       57,607         783,248        6,493,374
 353        840,854       51,406         789,448        5,703,925
 354        840,854       45,156         795,698        4,908,227
 355        840,854       38,857         801,997        4,106,230
 356        840,854       32,508         808,347        3,297,883
 357        840,854       26,108         814,746        2,483,137
 358        840,854       19,658         821,196        1,661,941
 359        840,854       13,157         827,697          834,244
 360        840,848        6,604         834,244                0

Premere un tasto per continuare
```

Figura 10.2 *Ammortamento dopo 30 anni*

Si noti che la formula usata nel programma per determinare il pagamento mensile è accurata. Tuttavia, dato che l'unità più piccola è una lira, è il sistema valutario stesso che introduce imprecisione. Non si possono pagare frazioni di lire e l'arrotondamento del pagamento mensile alla lira più vicina rende meno preciso il piano di ammortamento. Questa imprecisione viene compensata modificando l'ultima rata, in modo che il debito sia pienamente estinto.

GRAFICI A TORTA

I dati risultano molto più incisivi quando vengono rappresentati graficamente. Questa sezione presenta una programma generico per la creazione di grafici a torta, programma che è stato progettato per illustrare alcune tecniche di programmazione e per essere flessibile.

I grafici a torta sono ideali per mostrare il contributo (in percentuale) di determinati valori su un intero. Generalmente, al fine di rendere il grafico chiaro e leggibile, non si dovrebbero utilizzare più di 10 serie di valori in un diagramma a torta; per un maggior numero di serie si dovrebbe utilizzare un grafico a barre, illustrato nella sezione successiva. Il Programma 10.4 consente di tracciare un grafico a torta che comprenda fino a 10 diverse categorie. L'unica modifica necessaria per cambiare il

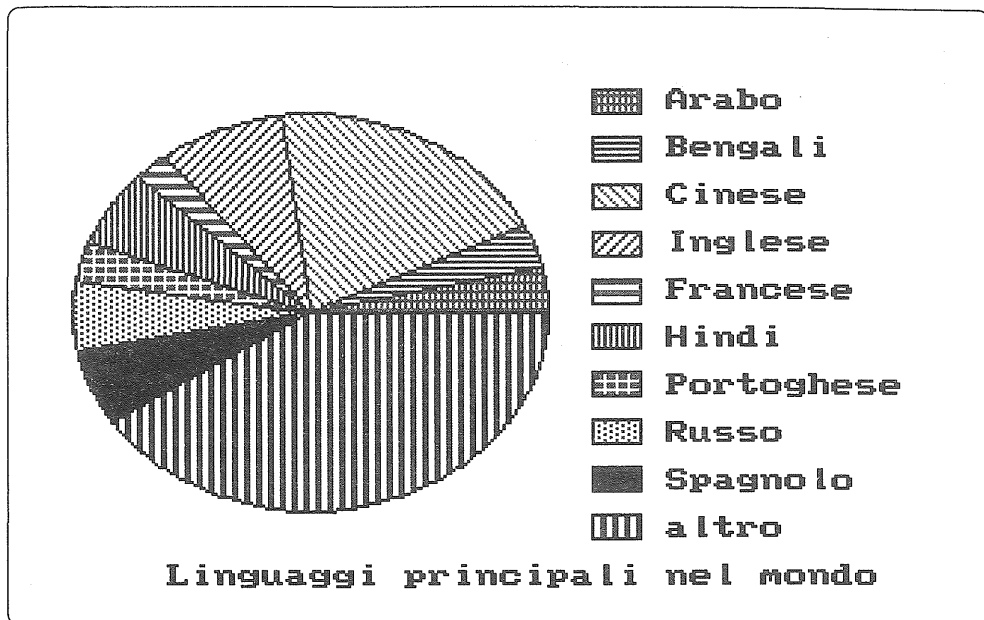



Figura 10.3 *Un grafico a torta generato dal Programma 10.4*

grafico visualizzato, è la sostituzione dei numeri e delle rispettive categorie che si trovano negli enunciati DATA all'inizio del programma. Se i nomi delle singole categorie fossero molto lunghi, si può ridurre la dimensione del grafico decrementando il valore relativo alla scala.

La Figura 10.3 mostra l'output del Programma 10.4. La figura rappresenta, in percentuale, la diffusione delle lingue nel mondo. Sullo schermo, i settori del cerchio appaiono a colori, e ad ogni settore è assegnato un motivo di riempimento differente. Per cambiare la tassellatura, si veda il Capitolo 8 per le informazioni relative ai motivi di riempimento, e si modifichino i caratteri a destra del segno uguale nel sottoprogramma AssignTilingPatterns.

In un grafico a torta, ogni settore (o fetta) rappresenta una percentuale dell'intera torta. Quindi, il programma deve convertire i dati numerici in percentuali. Nel Programma 10.4, il sottoprogramma CountAndTotalData conta il numero di inserimenti differenti nel primo enunciato DATA e totalizza questi numeri.

Nota: L'ultimo valore in questo enunciato DATA, -1, è un valore sentinella che segnala la fine dei dati.

La variabile count fornisce il numero di settori che devono apparire nel grafico a torta, mentre la variabile *total* viene usata per convertire ogni numero del primo enunciato DATA in percentuali.

Il sottoprogramma DrawChart traccia e riempie i settori del diagramma a torta. Per capire questo sottoprogramma nono necessarie alcune informazioni addizionali sull'enunciato CIRCLE.

La Figura 10.4 mostra un cerchio diviso da alcuni raggi. Il raggio che si estende dal centro del cerchio verso destra è chiamato raggio orizzontale. Ad ogni raggio è assegnato un numero compreso tra 0 e 1 che fornisce la percentuale del cerchio tra un determinato raggio e il raggio orizzontale. La misura in radianti dell'angolo compreso tra il raggio orizzontale e un qualsiasi altro raggio, è uguale a 2*pi moltiplicato per il numero assegnato a quel raggio. L'enunciato

```
CIRCLE (x,y), r, , -r1, -r2
```

traccia il settore del cerchio di raggio *r* limitato dal raggio che forma gli angoli di *r1* e *r2* radianti con il raggio orizzontale. Il settore inizia al raggio associato a *r1* e termina con il raggio associato a *r2*. Questo settore contiene la porzione (*r2-r1*)/(2*pi) dell'area dell'intero cerchio. In altre parole, partendo da *r1*, la porzione *p* dell'area del cerchio è contenuta nel settore compreso tra *r1* e *r2*, dove *r2* è uguale a *r1*+2*pi**p*.

Il Programma 10.4 crea un grafico a torta, un settore alla volta, iniziando dal settore che si trova sopra al raggio orizzontale. Per questo settore, *r1* è uguale a zero e l'enunciato CIRCLE comporta l'uso del valore -0. Dato che il computer non è in grado di distinguere 0 da -0, non genera il risultato voluto. Per rimediare a questa situazione, viene usato un numero molto piccolo, come -0,00001, invece di -0. Il programma traccia quindi i settori del grafico a torta uno dopo l'altro e utilizza il raggio finale di ciascun settore come inizio di quello successivo.

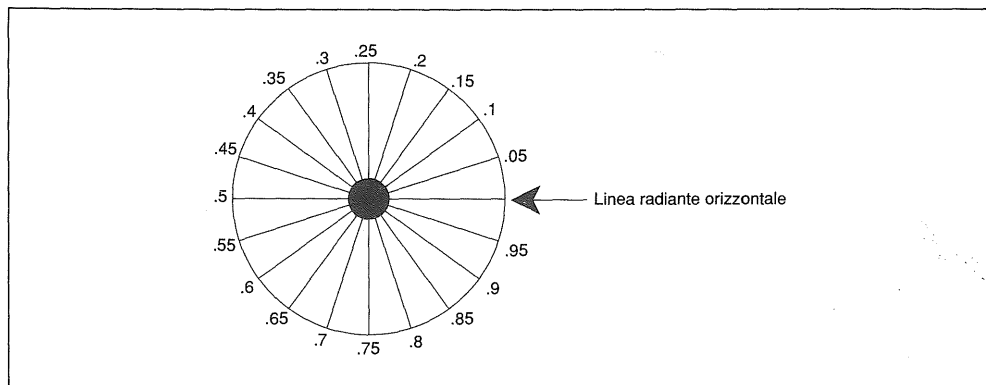



Figura 10.4 *Le porzioni di un cerchio*

Nel Programma 10.4, l'enunciato PAINT, usato per riempire un settore con un motivo, richiede le coordinate di un punto contenuto nel settore. Un metodo per trovare questo punto è quello di selezionare il punto in mezzo del raggio divide il settore. Se il settore è limitato dal raggio la cui misura in radianti corrisponde a $r1$ e $r2$, la misura associata al raggio di divisione è $t=(r1+r2)/2$. Si possono usare le funzioni trigonometriche per calcolare il punto centrale di questo raggio. Le coordinate del punto sono $x+(r/2)$*COS(t) e $y+(r/2)$*SIN(t), dove (x,y) costituisce il centro del cerchio e r il raggio.

C'è un'ultima considerazione da fare. Si deve usare un enunciato WINDOW per specificare un sistema di coordinate. Dopo questa operazione, il cerchio tracciato dal comando CIRCLE (x,y), r avrà un raggio orizzontale di lunghezza r. Il raggio verticale apparirà sullo schermo con la stessa lunghezza. Tuttavia, la lunghezza impostata da WINDOW in base alla scala y sarà molto probabilmente diversa da r. Perché le funzioni trigonometriche appena fornite possa funzionare correttamente, è necessario che queste due lunghezze corrispondano. A questo scopo, l'enunciato WINDOW deve specificare la lunghezza dell'asse x e dell'asse y con un rapporto di 4 a 3. Ciò è dovuto al fatto che la lunghezza dei due lati dello schermo sono in rapporto 4 a 3. Un enunciato appropriato è WINDOW (0,0)-(4,3).

Programma 10.4 *Creazione di un grafico a torta*

```
REM Creazione di un diagramma a torta  [10-4]
scale = 1
'La dimensione della torta può essere cambiata modificando il
'valore della scala. Al diminuire della scala, aumenta lo
'lo spazio per le descrizioni. Una scala di 1 assegna un
'alla torta un diametro uguale a metà larghezza dello
'schermo
DIM tile$(1 TO 10)
CALL AssignTilingPatterns(tile$())
CALL CountAndTotalData(itemCount, sumOfData)
CALL DrawChart(itemCount, sumOfData, tile$(), scale)
CALL DisplayLegend(itemCount, tile$(), scale)
' Dati: numero di persone (in milioni) che parlano una lingua
DATA 177, 171, 974, 420, 114, 300, 164, 285, 296, 2000, -1
REM -- Dati: i linguaggi più importanti
DATA Arabo, Bengali, Cinese, Inglese, Francese
DATA Hindi, Portoghese, Russo, Spagnolo, altro
REM -- Dati: titolo per il grafico a torta
DATA Linguaggi principali nel mondo
END
```

```
SUB AssignTilingPatterns (tile$())
  'motivi per le fette della torta
  tile$(1) = CHR$(136) + CHR$(136) + CHR$(170)
  tile$(2) = CHR$(85) + CHR$(0)
  tile$(3) = CHR$(128) + CHR$(32) + CHR$(8) + CHR$(2)
  tile$(4) = CHR$(3) + CHR$(12) + CHR$(48) + CHR$(192)
  tile$(5) = CHR$(170) + CHR$(170) + CHR$(0) + CHR$(0)
  tile$(6) = CHR$(17)
  tile$(7) = CHR$(168) + CHR$(168) + CHR$(0)
  tile$(8) = CHR$(1) + CHR$(16)
  tile$(9) = CHR$(255)
  tile$(10) = CHR$(5)
END SUB

SUB CountAndTotalData (count, total)
  count = 0
  total = 0
  READ itemValue
  DO UNTIL itemValue < 0
    total = total + itemValue
    count = count + 1
    READ itemValue
  LOOP
END SUB

SUB DisplayLegend (count, tile$(), scale)
  heightInRows = 25
  widthInColumns = 40
  WINDOW SCREEN (0, 0)-(widthInColumns, heightInRows)
  boxWidth = 2
  boxHt = 1
  legendTop = (heightInRows - (boxHt + 1) * count) \ 2
  leftEdge = widthInColumns * (scale / 2) + 2
  FOR i = 1 TO count
    READ itemName$
    boxBottom = legendTop + (boxHt + 1) * i
    boxTop = boxBottom - boxHt
    rtEdge = leftEdge + boxWidth
    LINE (leftEdge, boxTop)-(rtEdge, boxBottom), , B
    PAINT (leftEdge + boxWidth / 2, boxBottom - boxHt / 2), tile$(i)
    LOCATE boxBottom, leftEdge + boxWidth + 2: PRINT itemName$
  NEXT i
  'display title
  READ title$
  LOCATE heightInRows - 1, (widthInColumns - LEN(title$)) / 2
  PRINT title$;
END SUB
```

```
SUB DrawChart (count, total, tile$(), scale)
  'I risultati sono indipendenti dai valori di
  'windowWidth e windowHeight. Si tenga presente
  'che questi valori sono in rapporto 4 a 3.
  windowWidth = 4
  windowHeight = 3
  SCREEN 1, 0
  COLOR , 0
  WINDOW (0, 0)-(windowWidth, windowHeight)
  twoPi = 8 * ATN(1)              '2*pi
  radius = scale * windowWidth / 4
  xcenter = radius
  ycenter = windowHeight / 2
  startSector = .00001
  RESTORE   'usa ancora il primo enunciato DATA
  FOR index% = 1 TO count
    READ itemValue
    endSector = startSector + twoPi * (itemValue / total)
    IF endSector > 6.283 THEN endSector = 6.283
    CIRCLE (xcenter, ycenter), radius, , -startSector, -endSector
    theta = (startSector + endSector) / 2
    x = xcenter + radius * COS(theta) / 2
    y = ycenter + radius * SIN(theta) / 2
    PAINT (x, y), tile$(index%)
    startSector = endSector
  NEXT index%
  READ itemValue  'legge il valore sentinella
END SUB
```

Dopo aver tracciato il grafico a torta, il sottoprogramma DisplayLegend visualizza le legende necessarie per identificare i motivi di riempimento associati a ciascuna categoria. Questo sottoprogramma traccia dei riquadri rettangolari usando il comando LINE, e li riempie tramite l'enunciato PAINT. I rettangoli vengono quindi centrati verticalmente sul lato di destra dello schermo. Dopo aver visualizzato le legende, il Programma 10.4 mostra il titolo del grafico a torta, centrato nella parte inferiore dello schermo.

GRAFICI A BARRE

Un altro tipo di grafico comunemente usato per rappresentare i dati, è il grafico a barre. In questa sezione viene presentato un programma che visualizza un grafico a barre sullo schermo e produce una stampa ad alta qualità.

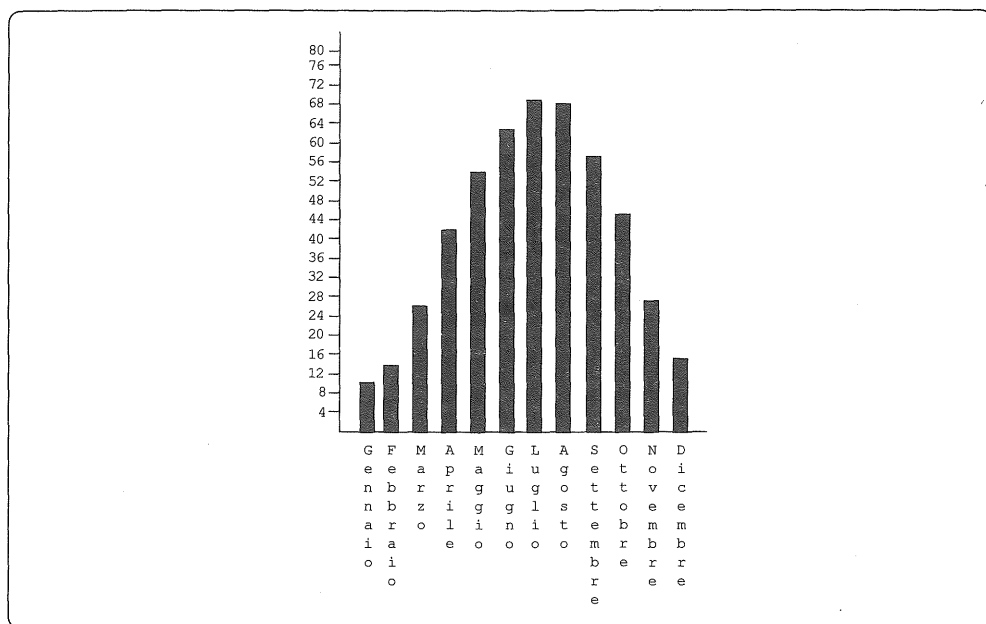



Figura 10.5 *Un grafico a barre creato dal Programma 10.8*

La Figura 10.5 mostra una stampa ottenuta dal Programma 10.8.

Seguono alcune delle caratteristiche di questo grafico:

- le lettere assomigliano più ai caratteri generati da una stampante in modalità testo che a quelli prodotti da una stampante in modalità grafica;
- i trattini sull'asse y sono equidistanti e allineati alle loro etichette (ognuna di queste etichette può contenere fino a 4 cifre);
- le etichette sotto le barre sono centrate e scritte in verticale (sullo schermo, le etichette possono essere composte al massimo da tre caratteri; tuttavia, non c'è limite al numero di caratteri per etichetta nella versione stampata).

A questo punto, si analizzeranno tutte queste operazioni per capire il funzionamento del Programma 10.8.

IMPOSTAZIONE DI UN SISTEMA DI COORDINATE, DEGLI ASSI E DEI TRATTINI

La prima operazione da svolgere in un qualsiasi programma grafico consiste nel selezionare una modalità SCREEN appropriata e un sistema di coordinate. La modalità ad alta risoluzione risulta più adatta ai grafici a barre, soprattutto perché utilizza dei caratteri più piccoli. Ciò riduce al minimo la quantità di spazio richiesta dalle etichette, un aspetto molto importante nella formattazione dei grafici a barre. L'enunciato

```
WINDOW (-40,-27)-(599,172)
```

imposta un sistema di coordinate appropriato. Dato che lo schermo, in alta risoluzione, è diviso in 640x200 pixel, ogni pixel corrisponde a un punto con coordinate intere; ciò significa che un'unità in questo sistema di coordinate è ha larghezza e altezza uguale a un pixel.

La Figura 10.6 spiega perché sono stati usati i valori -40 e -27 nell'enunciato WINDOW. Dato che ogni carattere è composto, sia in altezza che in larghezza, da 8 pixel, esiste uno spazio per quattro cifre e un trattino a sinistra dell'asse y, e uno spazio per tre caratteri sotto all'asse x. Inserendo l'asse x nel ventottesimo pixel (partendo dal basso), questo viene centrato verticalmente nella ventiduesima riga di testo. Se si parte dall'asse x e ci si sposta di 8 pixel per ogni trattino successivo, ci si assicura che ogni trattino venga centrato verticalmente in una riga di testo. Quindi, anche se QBasic traccia i trattini tramite enunciati grafici e visualizza le etichette come testo, i trattini e le etichette saranno sempre perfettamente allineate.

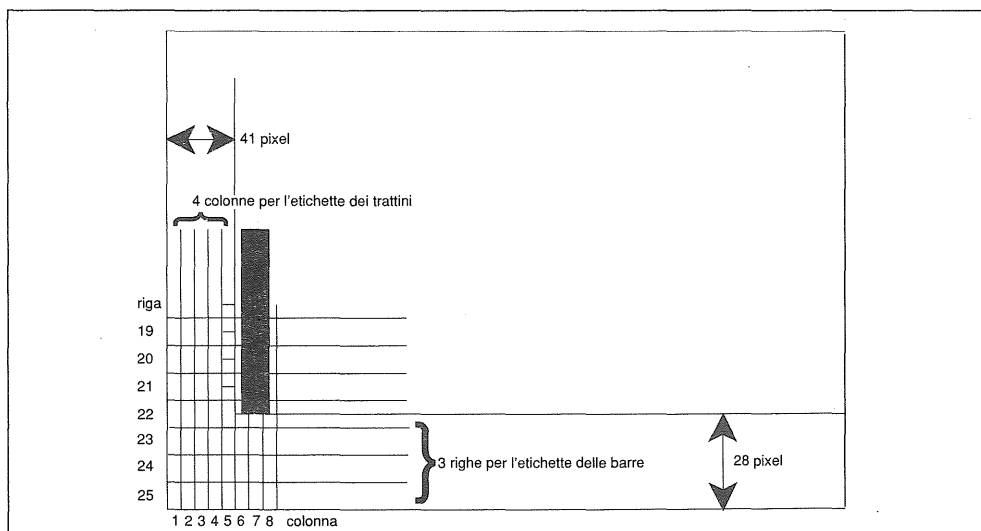



Figura 10.6 *Scelta delle coordinate per un grafico a barre*

Gli enunciati

```
LINE (0,0)-(599,0)
LINE (0,0)-(0,164)
```

tracciano gli assi. L'asse y si estende verso l'alto per soli 164 pixel, al fine di consentire al programma di utilizzare gli 8 pixel superiori per visualizzare il titolo. Infine, gli enunciati

```
FOR verticalPosition% = 8 TO 160 STEP 8
   LINE (0,verticalPosition%)-(-8,verticalPosition%)
NEXT verticalPosition%
```

aggiungono 20 trattini (ogni coppia separata da 8 pixel) a sinistra dell'asse y.

DETERMINAZIONE DELLA SCALA DELL'ASSE Y

I numeri usati come etichette per i trattini sull'asse y dipendono dal valore rappresentato dalla barra più alta. Ci si riferirà a questo valore come maximumDataValue. Un modo per determinare la scala consiste nell'etichettare il trattino più alto con questo valore e il più basso con 1/20 di questo valore. Un altro metodo consiste nell'etichettare il trattino più basso con un valore arrotondato di maximumDataValue/20, ed etichettare gli altri di conseguenza. Il primo metodo potrebbe generare delle etichette inconsuete. Benché il secondo metodo sembri più ragionevole, si preferisce in genere utilizzare delle etichette la cui distanza ha incrementi di 5, 10 o 100. Il Segmento di programma 10.5 seleziona dei valori per le etichette associate ai trattini scegliendo un numero ragionevole come intervallo. Questo segmento di programma specifica inoltre un formato di visualizzazione tramite il comando PRINT USING.

VISUALIZZAZIONE DELLE ETICHETTE IN VERTICALE

Le etichette visualizzate sotto alle barre potrebbero essere inserite orizzontalmente se le barre fosse sufficientemente larghe e le etichette abbastanza corte. Tuttavia, si ottiene una maggiore flessibilità se si visualizzano le etichette verticalmente, sotto ogni barra. Questo formato consente di centrare le barre rispetto alla posizione dei caratteri sottostanti, e di stampare su carta etichette di qualsiasi dimensione. Come detto in precedenza, il programma visualizza solo i primi tre caratteri di ciascuna etichetta se l'output è indirizzato sullo schermo.

Se le etichette di ciascuna barra sono contenute nell'array denominato *name$()* e si aggiungono successivamente delle barre in modo che siano centrate rispetto alle colonne 7, 9, 11 e così via, le etichette possono essere visualizzate con gli enunciati del Segmento di programma 10.6. In questo caso, viene usata la funzione MID$ per

selezionare la lettera *letter%* da ciascuna etichetta. Se non esiste una lettera *letter%* in una determinata etichetta, MID$ restituisce una stringa nulla. Per mantenere corretta la spaziatura per le altre etichette, il programma converte questo valore nullo in uno spazio.

Segmento di programma 10.5 *Impostazione dell'intervallo della scala*

```
s = maximumDataValue / 20
SELECT CASE s
  CASE IS >= 100
    interval = Ceil(s / 100) * 100
    format$ = "####"
  CASE IS >= 10
    interval = Ceil(s / 10) * 10
    format$ = "####"
  CASE IS >= 1
    interval = Ceil(s / 1) * 1
    format$ = "####"
  CASE IS >= .1
    interval = Ceil(s * 10) / 10
    format$ = "##.#"
  CASE IS >= .01
    interval = Ceil(s * 100) / 100
    format$ = "#.##"
  CASE ELSE
    interval = Ceil(s * 1000) / 1000
    format$ = ".###"
END SELECT
FOR tick% = 1 TO 20
  LOCATE 22 - tick%, 1
  PRINT USING format$; interval * tick%;
NEXT tick%

FUNCTION Ceil (x)
  REM L'intero più piccolo maggiore o uguale a x
  Ceil = -INT(-x)
END FUNCTION
```

Segmento 10.6 *Visualizzazione di etichette in verticale*

```
FOR bar% = 1 TO barCount%
  FOR letter% = 1 TO 3
    LOCATE 22 + letter%, 7 + 2 * (bar% - 1)
    letter$ = MID$(names$(bar%), letter%, 1)
    IF letter$ = "" THEN letter$ = " "
    PRINT letter$;
  NEXT letter%
NEXT bar%
```

POSIZIONAMENTO E VISUALIZZAZIONE DELLE BARRE

L'altezza di ciascuna barra riflette il valore rappresentato dalla barra stessa. Dato che l'altezza di un singolo pixel è *intervallo*/8, l'altezza di una barra (in pixel) equivale a

```
valore rappresentato dalla barra / (intervallo/8)
```

Dato che le etichette sotto le barre sono separate da un solo spazio, la larghezza massima di ciascuna barra può essere di 16 pixel. Quindi, la prima barra dovrebbe iniziare al quarto pixel dell'asse x ed estendersi fino al diciannovesimo pixel. La seconda barra dovrebbe estendersi dal ventesimo a trentacinquesimo pixel, e così via. Le barre possono quindi essere tracciate dagli enunciati del Segmento di programma 10.7, che presume che le etichette e i valori delle barre siano memorizzati, rispettivamente, nell'array *name$()* e *values()*. Il parametro BF nell'enunciato LINE traccia un rettangolo pieno i cui due angoli opposti vengono specificati dall'utente.

Le barre tracciate dal Segmento di programma 10.7 sono corrette, ma si toccano. Il grafico risulterebbe meglio interpretabile se ci fosse dello spazio tra le barre. Quindi, l'enunciato LINE dovrebbe essere sostituito da

```
LINE (leftEdge%+1,0)-(leftEdge%+barWidth%-1,height%), , BF
```

Segmento di programma 10.7 *Disegno di due barre*

```
leftEdge% = 4
barWidth% = 16
FOR bar% = 1 TO barCount%
  height% = values(bar%) / (interval / 8)
  LINE (leftEdge%, 0)-(leftEdge% + barWidth% - 1, height%), , BF
  leftEdge% = leftEdge% + barWidth%
NEXT bar%
```

STAMPA DELLE BARRE SU UNA STAMPANTE A MATRICE DI PUNTI

Il metodo standard per stampare il contenuto dello schermo prevede l'uso di un apposito programma. Il DOS mette a disposizione il programma denominato GRAPHICS.COM. Dopo aver caricato in memoria questo programma dal prompt del DOS, si può premere il tasto Stamp per inviare il contenuto dello schermo alla stampante attiva. Tuttavia, questo metodo comprende alcune limitazioni. Innanzi tutto, risulta piuttosto lento, soprattutto quando un grafico contiene poche barre e la maggior parte dello schermo è vuoto. In secondo luogo, l'immagine stampata è distorta rispetto a quella visualizzata sullo schermo.

Esiste un altro metodo per stampare un grafico a barre che non solo scavalca le limitazioni imposte dalla stampa effettuata con il tasto Stamp, ma consente di stampare tutte le lettere delle etichette. Questo metodo comporta la scrittura di una serie di routine personalizzate che utilizzino i dati originali per rappresentare graficamente i dati direttamente sulla stampante. I sottoprogrammi di stampa presentati nel Programma 10.8 costituiscono un esempio di una serie di routine di questo tipo. Il sottoprogramma di stampa principale, PrintChart, richiama inizialmente PrepareToPrint per impostare la stampante per la grafica, stampa il titolo del grafico e utilizza quindi il sottoprogramma PrintLine per generare il corpo del diagramma. Viene quindi richiamato il sottoprogramma PrintLastLine che riproduce l'asse orizzontale e le porzioni delle barre che si trovano sopra quest'asse. Infine, il programma richiama PrintBarLabels per stampare, in colonne verticali, tutte le etichette.

I sottoprogrammi di stampa del Programma 10.8 usano una scala di *intervallo*/16 al fine di conferire al grafico stampato la stessa forma di quello visualizzato. Come risultato, ogni riga visualizzata corrisponde a due righe stampate. Quando i sottoprogrammi devono stampare le righe 'in eccesso', stampano solamente una porzione dell'asse y e una parte di ciascuna barra, in modo da assicurare continuità nella stampa del grafico.

Quando si stampa in modalità standard, non si possono stampare più di 80 caratteri in una singola riga, e le righe hanno una determinata spaziatura. In modalità grafica, invece, si possono stampare quasi 1000 caratteri per riga per formare un'immagine larga 8 pollici. Inoltre, bisogna eliminare la spaziatura tra le righe. Nel Programma 10.8, il sottoprogramma PrepareToPrint imposta la larghezza di stampa su 255, il massimo consentito da QBasic, in modo che non venga aggiunto un carattere di ritorno a capo e avanzamento riga prima che tutti i caratteri grafici per una determinata riga siano stati inviati alla stampante.

PrepareToPrint, inoltre, imposta una spaziatura di 8/72 di pollice al fine di rimuovere la spaziatura tra le righe. PrepareToPrint fornisce anche le stringhe che definiscono i tre caratteri grafici usati nella stampa dell'asse verticale.

Il sottoprogramma PrintLine gestisce la stampa di una determinata riga. PrintLine stampa le etichette associate a trattini, o quattro spazi, seguite da una stringa di dati grafici. I dati grafici relativi a una determinata riga iniziano con un trattino e una porzione dell'asse verticale, o solo una porzione dell'asse verticale. Questi dati sono seguiti da una serie di motivi di riempimento, uno per ciascuna barra nel grafico.

La funzione Pattern% determina i motivi di riempimento da stampare per ciascuna barra, nonché la quantità della barra contenuta nella riga di stampa corrente. Se una determinata barra si estende sopra la riga di stampa corrente, il valore della funzione è 255, che indica al programma di stampare un rettangolo pieno quando la testina di stampa raggiunge la posizione della barra. Se una barra non raggiunge la riga di stampa corrente, il valore della funzione è 0, che indica di non stampare nulla quando

la testina di stampa raggiunge la posizione della barra. Quando una barra si estende parzialmente nella riga di stampa corrente, la funzione restituisce un valore compreso tra 1 e 127, che indica al programma la porzione di rettangolo appropriata che deve essere stampata quando la testina di stampa raggiunge la posizione della barra.

CREAZIONE E STAMPA DEI GRAFICI A BARRE

Il Programma 10.8 include tutte le caratteristiche discusse in queste sezioni. Il programma consente di riprodurre un massimo di 37 barre utilizzando i valori forniti interattivamente dall'utente durante l'esecuzione. Sono stati riservati quattro spazi per le etichette associate ai trattini. Questa impostazione non limita la grandezza dei valori da rappresentare. Ad esempio, se i valori fossero nell'ordine delle centinaia di migliaia, si potrebbero tralasciare le ultime tre cifre e aggiungere la parola migliaia al titolo del grafico.

GESTIONE DI DATABASE

Teoricamente, si potrebbe chiamare database una qualsiasi collezione di dati. Tuttavia, il termine database si riferisce normalmente a una collezione di dati memorizzati su disco che possano assumere diversi aspetti a seconda delle esigenze del momento. Quindi, un database può servire come fonte di dati per diverse applicazioni.

Programma 10.8 *Creazione e stampa di un grafico a barre*

```
REM Visualizza e stampa un grafico a barre  [10-8]
CONST maxBars = 37
DIM values(1 TO maxBars), names$(1 TO maxBars)
'Questi sottoprogrammi condividono i valori variabili()
'names$(), title$, interval, maxLength%, e barCount%
CALL GetInfo
CALL PrepareScreenAxisAndTickMarks
CALL DrawBars
CALL LabelBars
LOCATE 1, 1: PRINT title$;
LOCATE 1, 50: PRINT "Vuoi stampare il grafico (S/N)?"
IF UCASE$(INPUT$(1)) = "S" THEN CALL PrintChart
END

FUNCTION Ceil (x)
  Ceil = -INT(-x)
END FUNCTION
```

```
SUB DrawBars
  SHARED values(), interval, barCount%
  leftEdge% = 4
  barWidth% = 16
  FOR bar% = 1 TO barCount%
    height% = values(bar%) / (interval / 8)
    LINE (leftEdge% + 1, 0)-(leftEdge% + barWidth% - 1, height%), , BF
    leftEdge% = leftEdge% + barWidth%
  NEXT bar%
END SUB

SUB FindNiceInterval (s)
  SHARED interval, format$
  SELECT CASE s
    CASE IS >= 100
      interval = Ceil(s / 100) * 100
      format$ = "####"
    CASE IS >= 10
      interval = Ceil(s / 10) * 10
      format$ = "####"
    CASE IS >= 1
      interval = Ceil(s / 1) * 1
      format$ = "####"
    CASE IS >= .1
      interval = Ceil(s * 10) / 10
      format$ = "##.#"
    CASE IS >= .01
      interval = Ceil(s * 100) / 100
      format$ = "#.##"
    CASE ELSE
      interval = Ceil(s * 1000) / 1000
      format$ = ".###"
  END SELECT
END SUB


SUB GetInfo
  SHARED values(), names$(), title$, interval, maxLength%, barCount%
  maxValue = 0
  maxLength% = 0
  barCount% = 0
  CLS
  CALL GiveInstructions
  INPUT "Nome"; newName$
  DO WHILE newName$ <> ""
    IF LEN(newName$) > maxLength% THEN maxLength% = LEN(newName$)
    INPUT "Valore"; newValue
    IF newValue > maxValue THEN maxValue = newValue
    barCount% = barCount% + 1
    values(barCount%) = newValue
```

```
      names$(barCount%) = newName$
      IF barCount% = maxBars THEN EXIT DO
      'Rivisualizza le istruzioni in fondo allo schermo
      'quando spariscono dalla visualizzazione.
      IF barCount% MOD 10 = 0 THEN CALL GiveInstructions
      INPUT "Nome"; newName$
    LOOP
    LINE INPUT "Titolo? "; title$
    'Calcola un valore per l'intervallo verticale tra i simboli
    'che dovranno accomodare il valore più alto dei dati
    CALL FindNiceInterval(maxValue / 20)
  END SUB

  SUB GiveInstructions
    PRINT "Si possono fornire fino a"; maxBars;
    PRINT "nami e valori corrispondenti."
    PRINT "Una volta inseriti tutti i nomi e i valori,"
    PRINT "si prema semplicemente INVIO senza specificare"
    PRINT "un nome."
  END SUB

  SUB LabelBars
    SHARED barCount%, names$()
    FOR bar% = 1 TO barCount%
      FOR letter% = 1 TO 3
        LOCATE 22 + letter%, 7 + 2 * (bar% - 1)
        letter$ = MID$(names$(bar%), letter%, 1)
        IF letter$ = "" THEN letter$ = " "
        PRINT letter$;
      NEXT letter%
    NEXT bar%
  END SUB


  FUNCTION Pattern% (mode$, index%, bar%)
    SHARED values(), interval
    IF mode$ = "tick" THEN adjustment% = 4 ELSE adjustment% = 12
    extra% = values(bar%)/(interval/16) - 16*index% + adjustment%
    IF extra% < 0 THEN extra% = 0
    IF extra% >= 8 THEN
         Pattern% = 255
      ELSE
        Pattern% = 2 ^ (extra%) - 1
    END IF
  END FUNCTION

  SUB PrepareScreenAxisAndTickMarks
    SHARED interval, barCount%, format$
    SCREEN 2
    WINDOW (-40, -27)-(599, 172)
```

```
  LINE (0, 0)-(599, 0)
  LINE (0, 0)-(0, 164)
  FOR verticalPosition% = 8 TO 160 STEP 8
    LINE (0, verticalPosition%)-(-8, verticalPosition%)
  NEXT verticalPosition%
  FOR tick% = 1 TO 20
    LOCATE 22 - tick%, 1
    PRINT USING format$; interval * tick%;
  NEXT tick%
END SUB

SUB PrepareToPrint
  SHARED tick$, bar$, corner$
  WIDTH "LPT1:", 255
  'Imposta la stampante in grafica (8/72 di pollice)
  'Vengono forniti sia i codici IBM che quelli EPSON
  LPRINT CHR$(27); "@"; CHR$(27); "A"; CHR$(8); CHR$(2);
  LPRINT CHR$(27); "A"; CHR$(8);
  'Definisce un carattere da usare per l'asse verticale
  tick$ = STRING$(10,CHR$(8))+STRING$(2,CHR$(255))+STRING$(6,CHR$(0))
  bar$ = STRING$(10,CHR$(0))+STRING$(2,CHR$(255))+STRING$(6,CHR$(0))
  corner$ =
STRING$(10,CHR$(0))+STRING$(2,CHR$(248))+STRING$(6,CHR$(8))
END SUB

SUB PrintBarLabels
  SHARED names$(), maxLength%, barCount%
  FOR letterCount% = 1 TO maxLength%
    LPRINT "      ";
    FOR bar% = 1 TO barCount%
      letter$ = MID$(names$(bar%), letterCount%, 1)
      IF letter$ = "" THEN letter$ = " "
      LPRINT letter$; " ";
    NEXT bar%
    LPRINT
  NEXT letterCount%
END SUB


SUB PrintChart
  SHARED title$
  CALL PrepareToPrint
  LPRINT title$
  LPRINT
  'Stampa due righe per ogni riga dello schermo
  FOR index% = 20 TO 1 STEP -1
    CALL PrintLine("tick", index%)
    CALL PrintLine("bar", index%)
  NEXT index%
  CALL PrintLastLine
```

```
  CALL PrintBarLabels
END SUB

SUB PrintGraphics (graphicsString$)
  'Sequenze di escape per stampanti IBM, Epson, e cloni
  low% = LEN(graphicsString$) MOD 256
  high% = LEN(graphicsString$) \ 256
  LPRINT CHR$(27); "L"; CHR$(low%); CHR$(high%); graphicsString$;
END SUB

SUB PrintLastLine
  SHARED corner$, values(), barCount%, interval
  info$ = corner$
  FOR bar% = 1 TO barCount%
    extra% = values(bar%) / (interval / 16)
    IF extra% >= 4 THEN
        Ptn% = 248
      ELSE
        Ptn% = 2 ^ (extra% + 4) - 8
    END IF
    temp$ = STRING$(20, CHR$(Ptn%))
    info$ = info$ + CHR$(8) + CHR$(8) + temp$ + CHR$(8) + CHR$(8)
  NEXT bar%
  LPRINT "     ";
  CALL PrintGraphics(info$)
  LPRINT
END SUB



SUB PrintLine (mode$, index%)
  SHARED tick$, bar$, barCount%, interval, format$
  IF mode$ = "tick" THEN
      LPRINT USING format$; interval * index%;
      info$ = tick$
    ELSE
      LPRINT "    ";
      info$ = bar$
  END IF
  FOR bar% = 1 TO barCount%
    temp$ = STRING$(20, CHR$(Pattern%(mode$, index%, bar%)))
    info$ = info$ + CHR$(0) + CHR$(0) + temp$ + CHR$(0) + CHR$(0)
  NEXT bar%
  CALL PrintGraphics(info$)
  LPRINT
END SUB
```

Un programma di database è un programma in grado di creare e gestire un database.

Il Capitolo 7 ha introdotto i file ad accesso casuale, sequenziali e binari. In questa sezione, verranno utilizzati gli enunciati relativi alla gestione dei file per creare e gestire un database.

APERTURA DI UN DATABASE GENERICO

I dati principali di un database verranno memorizzati in un file ad accesso casuale. Per poter creare un file ad accesso casuale, bisogna decidere quanti campi ci devono essere in un record, e quale deve essere la lunghezza di ogni campo. Queste informazioni sono necessarie per poter utilizzare correttamente le istruzioni OPEN e FIELD. Inoltre, sapendo il tipo di dati che ciascun campo deve contenere, sarà più semplice assegnare dei nomi descrittivi alle variabili di campo. Ad esempio, si supponga che un database contenga dei nomi e dei numeri di telefono. Ogni record potrebbe essere composto dal campo nome, cognome e telefono di lunghezza, rispettivamente, di 10, 15 e 12. Con queste informazioni, i comandi appropriati per aprire il database sono

```
OPEN "PHONEDIR.DB" AS #1 LEN=37
FIELD #1, 10 AS firstNamef$, 15 AS lastNamef$, 12 AS phonef$
```

Per creare un programma per la gestione di un database, si dovrebbero generalizzare questi enunciati rimuovendo i valori specifici e sostituendoli con delle variabili. I comandi del Segmento di programma 10.9 mostrano questa generalizzazione. Si noti che gli enunciati OPEN e FIELD non contengono più delle costanti che sono specifiche per un database particolare. Queste due istruzioni possono aprire qualsiasi database con tre campi. Dato che le variabili di campo generiche *item1$, item2$* e *item3$* non sono più descrittive, il programma genera tre variabili addizionali, *item1Name$, item2Name$* e *item3$Name* per contenere una descrizione dei campi.

Si potrebbe continuare con questa generalizzazione introducendo degli array da usare come variabili di campo in cui memorizzare la lunghezza e i nomi dei campi . Inoltre, si possono inserire le informazioni relative a PHONEDIR.DB in alcuni enunciati DATA. Questi comandi per l'apertura di un database generico sono utilizzati nel Segmento di programma 10.10.

Questi enunciati sono così generali che sono sufficienti due piccole modifiche per applicarli a un database con 4, 5 o più campi. A questo scopo, si devono cambiare gli enunciati DATA alla fine del programma, e aggiungere più campi al comando FIELD. Si può eseguire automaticamente il comando FIELD appropriato per un determinato numero di campi (fino a un massimo numero predefinito) sostituendo l'enunciato FIELD nel Segmento di programma 10.10 con un'istruzione CALL che richiami la procedura SetFields riportata nel Segmento di programma 10.11. La procedura SetFields si serve di una struttura SELECT CASE per scegliere l'enunciato FIELD appropriato per il numero di campi specificato dalla variabile *fieldCount%.*

Segmento di programma 10.9 *Enunciati OPEN e FIELD generici*

```
fileName$ = "PHONEDIR.DB"
item1Len = 10
item1Name$ = "Nome"
item2Len = 15
item2Name$ = "Cognome"
item3Len = 12
item3Name$ = "Telefono"
recordLength% = item1Len + item2Len + item3Len
OPEN fileName$ AS #1 LEN = recordLength%
FIELD #1, item1Len AS item1f$, item2Len AS item2f$, _
          item3Len AS item3f$
```

Segmento di programma 10.10 *Apertura di un database usando enunciati DATA*

```
READ fileName$, fieldCount%
DIM itemName$(1 TO fieldCount%)   'descrizione del campo
DIM itemLen%(1 TO fieldCount%)    'lunghezza del campo
DIM itemf$(1 TO fieldCount%)      'variabile di campo
recordLength% = 0
FOR index% = 1 TO fieldCount%
  READ itemName$(index%), itemLen%(index%)
  recordLength% = recordLength% + itemLen%(index%)
NEXT index%
OPEN fileName$ FOR RANDOM AS #1 LEN = recordLength%
FIELD #1, itemLen%(1) AS itemf$(1), itemLen%(2) AS itemf$(2),_
          itemLen%(3) AS itemf$(3)
DATA "PHONEDIR.DB", 3
DATA "Nome", 10
DATA "Cognome", 15
DATA "Telefono", 12
```

La generalizzazione finale consiste nel rimuovere gli enunciati DATA dal programma, e inserire le informazioni che descrivono la struttura di un database in un file sequenziale. Per un determinato database, ci saranno quindi due file: un file sequenziale che contiene il numero, la descrizione e la lunghezza dei campi, e un file ad accesso casuale che contiene i record del database. In questa sezione si usa la convenzione di inserire la struttura del database in un file senza estensione e di memorizzare i record del database in un file con lo stesso nome ed estensione .DB.

Si supponga di aver creato un file sequenziale con i dati

numero campi
nome del primo campo
lunghezza del primo campo
nome del secondo campo
lunghezza del secondo campo

e così via. Questo file descrive un database generico. Gli enunciati nel Segmento di programma 10.12 aprono questo database.

Segmento di programma 10.11 *Procedura per la scelta di un enunciato FIELD*

```
SUB SetFields (fieldCount%)
  SELECT CASE fieldCount%
    CASE 1
      FIELD #1, itemLen%(1) AS itemf$(1)
    CASE 2
      FIELD #1, itemLen%(1) AS itemf$(1), itemLen%(2) AS itemf$(2)
    CASE 3
      FIELD #1, itemLen%(1) AS itemf$(1), itemLen%(2) AS itemf$(2),_
                itemLen%(3) AS itemf$(3)
    CASE 4
      FIELD #1, itemLen%(1) AS itemf$(1), itemLen%(2) AS itemf$(2),_
                itemLen%(3) AS itemf$(3), itemLen%(4) AS itemf$(4)
    'Si può continuare con questa serie di enunciato CASE
    'fino a raggiungere il numero massimo di campi permessi
  END SELECT
END SUB
```

Segmento di programma 10.12 *Apertura di un database usando un file di descrizione*

```
INPUT "Nome del database"; dataFile$
OPEN dataFile$ FOR INPUT AS #1
INPUT #1, fieldCount%
DIM itemf$(1 TO fieldCount%), itemLen%(1 TO fieldCount%),_
    itemName$(1 TO fieldCount%)
recordLength% = 0
FOR index% = 1 TO fieldCount%
  INPUT #1, itemName$(index%), itemLen%(index%)
  recordLength% = recordLength% + itemLen%(index%)
NEXT index%
CLOSE #1
OPEN dataFile$ + ".DB" FOR RANDOM AS #1 LEN = recordLength%
CALL SetFields(fieldCount%)
```

Dopo aver aperto un database generico, sono necessarie alcune operazioni standard per poterlo gestire. Queste operazioni includono:

- creazione di un nuovo database;
- aggiunta di un record alla fine di un database;
- inserimento di un record in mezzo a un database;
- cancellazione di un record

- visualizzazione dei record
- modifica di un record
- ordinamento dei record.

Le sezioni seguenti discutono dettagliatamente ciascuna di queste operazioni.

CREAZIONE DI UN NUOVO DATABASE

Quando si crea un database non sono necessari dei record, ma solo i nomi e le lunghezze dei campi. Gli enunciati interattivi del Segmento di programma 10.13, richiedono il nome di un nuovo database e i nomi e le lunghezze dei campi, e creano quindi un file sequenziale che contiene le informazioni relative alla struttura del database.

Nota: I comandi nel Segmento di programma 10.13 non aprono il database

Segmento di programma 10.13 *Creazione di un file di descrizione per un nuovo database*

```
CONST maxFields = 4
REDIM fieldName$(1 TO maxFields), fieldLen%(1 TO maxFields)
CLS
INPUT "Nome del database (massimo 8 lettere)"; dataFile$
PRINT
PRINT "Inserire un nome e una lunghezza di campo per riga."
PRINT "Separare il nome e la lunghezza dei campi con una virgola."
PRINT "Terminare la lista inserendo  ,0"
count% = 0     'Conta il numero di campi specificati
INPUT nom$, length%
DO UNTIL nom$ = "" OR length% = 0
  count% = count% + 1
  fieldName$(count%) = nom$
  fieldLen%(count%) = length%
  IF count% = maxFields THEN EXIT DO
  INPUT nom$, length%
LOOP
OPEN dataFile$ FOR OUTPUT AS #1
WRITE #1, count%
FOR index% = 1 TO count%
  WRITE #1, fieldName$(index%), fieldLen%(index%)
NEXT index%
CLOSE #1
ERASE fieldName$, fieldLen%
```

Gli enunciati memorizzano il numero di campi nella variabile *count%*. Si noti l'uso della costante denominata *maxFields*. Questa costante appare all'inizio del programma di gestione di database principale. Il valore di questa costante dovrebbe essere il numero che appare nell'ultimo enunciato CASE nel blocco SELECT CASE della procedura SetFields del programma principale.

AGGIUNTA DI UN RECORD ALLA FINE DEL DATABASE

L'aggiunta di un record vuoto alla fine del database è una delle operazioni più semplici nella gestione di un archivio. Innanzi tutto, bisogna determinare il numero dell'ultimo record del database. A questo scopo, si può usare il comando

```
lastRecord% = LOF(1) / recordLength%
```

Segmento di programma 10.14 *Aggiunta di un record vuoto in un database*

```
lastRecord% = LOF(1) / recordLength%
FOR index% = 1 TO fieldCount%
  LSET itemf$(index%) = ""
NEXT index%
PUT #1, lastRecord% + 1
```

La procedura consiste nell'assegnare una stringa nulla a ciascuna variabile campo e nell'inserire quindi questo record vuoto dopo l'ultimo record. Gli enunciati nel Segmento di programma 10.14 aggiungono un record vuoto a un database generico.

INSERIMENTO DI UN RECORD IN MEZZO A UN DATABASE

L'operazione per l'inserimento di un record in mezzo a un database è simile a quella appena esaminata, ma richiede che tutti i record compresi tra il punto in cui viene inserito il nuovo record e la fine del database, vengano spostati di una posizione. Si può effettuare questa operazione tramite un ciclo LOOP che prelevi (GET) i record e li inserisca (PUT) nella posizione successiva. Dato che lo spostamento del record *r* sovrascriverebbe il record *r*+1, il ciclo FOR inizia dall'ultimo record e procede all'indietro fino a raggiungere il record che occupa la posizione in cui si vuole effettuare l'inserimento. Come nella procedura del Segmento di programma 10.14, l'ultima operazione è quella di inserire un record vuoto nella posizione specificata. Se un database contiene molti record, un inserimento all'inizio dell'archivio richiederà un certo periodo di tempo. Gli enunciati nel Segmento di programma 10.15 inseriscono un record in un database generico.

CANCELLAZIONE DI UN RECORD DA UN DATABASE

Un metodo per cancellare un record da un database consiste nell'invertire la procedura per l'inserimento di un record. Se si volesse rimuovere il record *n%*, si potrebbero usare i comandi seguenti:

```
lastRecord% = LOF(1) / recordLength%
FOR index% = n% + 1 TO lastRecord%
   GET #1, index%
   PUT #1, index% - 1
NEXT index%
```

Segmento di programma 10.15 *Inserimento di un record in un database*

```
INPUT "Numero del record", n%
lastRecord% = LOF(1) / recordLength%
FOR index% = lastRecord% TO n% STEP -1
  GET #1, index%
  PUT #1, index% + 1
NEXT index%
FOR index% = 1 TO fieldCount%
  LSET itemf$(index%) = ""
NEXT index%
PUT #1, n%
```

Tuttavia, questi enunciati comportano due svantaggi. Innanzi tutto, come nel caso di un inserimento, se viene cancellato un record all'inizio del database potrebbe essere necessario un lungo periodo di tempo. In secondo luogo, lo spostamento verso l'alto di tutti i record genera un duplicato dell'ultimo record del database, *lastRecord%*, poiché questo viene copiato nella penultima posizione e non viene poi cancellato. Una soluzione possibile a questo problema consiste nel copiare tutti i record, ad eccezione di quello da cancellare, in un file temporaneo, cancellare il database originale, ed assegnare al file temporaneo il nome originale. Il Segmento di programma 10.16 cancella un record seguendo questa procedura.

Si noti che il Segmento di programma 10.16 esegue un nuovo comando FIELDS in modo che una singola variabile possa riferirsi a un intero record. Inoltre, al termine delle operazioni il database viene chiuso.

Dato che la cancellazione di un record richiede un certo periodo di tempo la copia dell'intero file, è pratica comune non cancellare fisicamente un record, ma contrassegnarlo come tale in modo che il programma lo ignori. In un secondo tempo, dopo aver contrassegnato diversi record per la cancellazione, il programma può copiare tutti i record non contrassegnati in un nuovo file ad accesso casuale e cancellare quello originale. Questa procedura è conosciuta come packing (compattazione). La selezione anticipata dei record da cancellare consente anche di recuperare dei record prima di eliminarli fisicamente dal disco.

Per poter contrassegnare un record per la cancellazione, è necessario un campo addizionale di un carattere all'inizio di ciascun record. Questo campo può essere creato automaticamente dal programma principale in fase di creazione di un nuovo database. A questo scopo, è sufficiente utilizzare un comando FIELD generico nella forma

```
FIELD #1, 1 AS statusf$, itemLen%(1) AS itemf$, itemLen%(2) AS itemf$(2),
eccetera
```

La procedura di cancellazione si riduce al caricamento del record specificato (GET), all'impostazione della variabile *statusf$* con un valore particolare, ad esempio un'asterisco, e alla riscrittura del record nel database (PUT). Il Segmento di programma 10.17 attiva e disattiva lo stato di cancellazione di un record.

La procedura di compattazione consiste nel copiare in un altro file tutti i record non contrassegnati da un asterisco. Il Segmento di programma 10.18 svolge questa operazione. Si noti che la variabile *recordLength%* include ora la lunghezza di *statusf$*. Al termine dell'operazione, il database viene chiuso.

Segmento di programma 10.16 *Cancellazione di un singolo record*

```
lastRecord% = LOF(1) / recordLength%
FIELD #1, recordLength% AS file1f$
OPEN "temp" FOR RANDOM AS #2 LEN = recordLength%
FIELD #2, recordLength% AS file2f$
'Copia tutti i record prima di n%
FOR index% = 1 TO n% - 1
  GET #1, index%
  LSET file2f$ = file1f$
  PUT #2, index%
NEXT index%
'Copia tutti i record dopo n%
FOR index% = n% + 1 TO lastRecord%
  GET #1, index%
  LSET file2f$ = file1f$
  PUT #2, index% - 1
NEXT index%
CLOSE #1, #2
KILL dataFile$ + ".DB"
NAME "temp" AS dataFile$ + ".DB"
```

VISUALIZZAZIONE DEI RECORD DI UN DATABASE

È possibile visualizzare tutti i record contenuti in un database, o solo quelli il cui numero cade in un intervallo specificato. Inoltre, l'elenco visualizzato può contenere tutti i campi, o solo alcuni di essi. Il Segmento di programma 10.19 visualizza il numero

di record e il contenuto di ciascun campo per tutti i record del database. Dopo la visualizzazione dei primi 15 record, il programma si ferma per consentire all'utente di esaminare la lista e prosegue visualizzando i 15 successivi dopo la pressione di un tasto qualsiasi.

Segmento di programma 10.17 *Selezione dei record da cancellare*

```
INPUT "Numero di record da cancellare/recuperare", n%
GET #1, n%
IF statusf$ = "*" THEN
    LSET statusf$ = " "
    PRINT "Record recuperato"
  ELSE
    LSET statusf$ = "*"
    PRINT "Record cancellato"
END IF
PUT #1, n%
```

Segmento di programma 10.18 *Compattazione di un database*

```
lastRecord% = LOF(1) / recordLength%
FIELD #1, 1 AS statusf$, recordLength% - 1 AS file1f$
OPEN "temp" FOR RANDOM AS 2 LEN = recordLength%
FIELD #2, recordLength% AS file2f$
tempIndex% = 1
FOR index% = 1 TO lastRecord%
  GET #1, index%
  IF statusf$ = " " THEN
    LSET file2f$ = statusf$ + file1f$
    PUT #2, tempIndex%
    tempIndex% = tempIndex% + 1
  END IF
NEXT index%
CLOSE #1, #2
KILL dataFile$ + ".DB"
NAME "temp" AS dataFile$ + ".DB"
```

Segmento di programma 10.19 *Visualizzazione dei record di un database*

```
lastRecord% = LOF(1) / recordLength%
FOR recordNumber% = 1 TO lastRecord%
  GET #1, recordNumber%
  PRINT USING "! #####"; statusf$; recordNumber%;
  FOR index% = 1 TO fieldCount%
    PRINT " "; itemf$(index%);
  NEXT index%
  PRINT
```

```
    IF (recordNumber% MOD 15 = 0) OR (recordNumber% = lastRecord%) THEN
        PRINT
        PRINT "Premere un tasto per continuare";
        a$ = INPUT$(1)
        PRINT
    END IF
NEXT recordNumber%
```

MODIFICA DI UN RECORD

Ci sono diversi metodi per modificare un record in un database. Il modo più semplice è quello illustrato dal Segmento di programma 10.20, che visualizza il nome e il contenuto di ogni campo (uno alla volta) del record specificato e richiede all'utente di apportare le modifiche desiderate. Se viene premuto Invio senza specificare un nuovo valore, il campo in questione non viene modificato.

Delle routine più sofisticate potrebbero visualizzare tutti i campi contemporaneamente e consentire all'utente di spostare il cursore nella posizione desiderata per apportare le modifiche del caso.

Segmento di programma 10.20 *Modifica di un record di un database*

```
INPUT "Record da modificare", n%
GET #1, n%
FOR index% = 1 TO fieldCount%
  PRINT itemName$(index%)
  PRINT itemf$(index%)
  INPUT "Nuovo valore", newItem$
  IF newItem$ <> "" THEN LSET itemf$(index%) = newItem$
NEXT index%
PUT #1, n%
```

ORDINAMENTO DEI RECORD IN BASE AL CONTENUTO DI UN CAMPO

La tecnica per l'ordinamento di un database è simile a quella esaminata nel Capitolo 7 per l'ordinamento di un file sequenziale. Se la quantità di memoria disponibile fosse sufficiente, si potrebbe caricare l'intero database in più array ed ordinare questi array seguendo la procedura descritta nel Capitolo 7. Tuttavia, per eseguire un ordinamento sulla base di un singolo campo, esiste un metodo più rapido ed efficiente che richiede solo due array paralleli. A questo scopo, si devono assegnare i valori del campo di ordinamento al primo array, e i numeri di record corrispondenti al secondo array. A questo punto, ogni elemento del secondo array contiene il numero di record

dell'elemento corrispondente del primo array. È ora sufficiente ordinare questi array paralleli sulla base degli elementi del primo array e copiare l'intero database in un file temporaneo seguendo l'ordine indicato dai numeri di record contenuti nel secondo array. La procedura di ordinamento viene completata cancellando il database precedente ed assegnando al file temporaneo il nome originale. Questa tecnica, conosciuta come ordinamento con puntatori, è illustrata nel Segmento di programma 10.21. Dopo aver ordinato un database, si può utilizzare una ricerca binaria per localizzare rapidamente dei record che contengono un determinato valore nel campo usato per l'ordinamento.

UN PROGRAMMA GENERICO PER LA GESTIONE DI DATABASE

Le sezioni precedenti hanno illustrato diverse tecniche per la scrittura di un programma che consenta di creare e gestire un qualsiasi tipo di database. Viene ora presentato un programma di questo tipo che utilizza le tecniche finora presentate e che risulta abbastanza flessibile per poter essere modificato a seconda delle esigenze personali.

Come si può intuire dalle istruzioni inserite all'inizio del programma, si possono specificare al massimo 10 campi per record (si può incrementare questo numero aggiungendo degli enunciati CASE nella procedura SetFields). Inoltre, il programma limita la lunghezza dei nomi di campo a 15 caratteri.

Segmento di programma 10.21 *Ordinamento di un database*

```
lastRecord% = LOF(1) / recordLength%
INPUT "Numero di campo su cui basare l'ordinamento", fieldNumber%
DIM recNum%(1 TO lastRecord%), value$(1 TO lastRecord%)
FOR index% = 1 TO lastRecord%
  GET #1, index%
  recNum%(index%) = index%
  value$(index%) = itemf$(fieldNumber%)
NEXT index%
CALL SortArrays(recNum%(), value$())
FIELD #1, 1 AS statusf$, recordLength% - 1 AS file1f$
OPEN "temp" FOR RANDOM AS #2 LEN = recordLength%
FIELD #2, recordLength% AS file2f$
FOR index% = 1 TO lastRecord%
  GET #1, recNum%(index%)
  LSET file2f$ = statusf$ + file1f$
  PUT #2, index%
NEXT index%
CLOSE #1, #2
KILL dataFile$ + ".DB"
NAME "temp" AS dataFile$ + ".DB"
```

Il programma visualizza un menu da cui è possibile selezionare una serie di comandi; per svolgere una determinata operazione, è sufficiente premere il tasto racchiuso tra i simboli < e >. Sono già state spiegate tutte le opzioni disponibili ad eccezione di quelle che consentono, rispettivamente, di visualizzare la struttura di un database (cioè i nomi e le lunghezze dei campi) e di visualizzare un elenco di file di database (cioè i file con estensione .DB).

Molte procedure condividono le variabili *dataFile$*, *fieldCount%*, *itemf$()*, *itemLen%()*, *itemName$()*, *recordLength%* e *lastRecord%*. Queste variabili contengono le informazioni seguenti sul database correntemente aperto:

- la variabile *dataFile$* contiene il nome del database. Se *dataFile$* è una stringa nulla, significa che non è stato aperto nessun database;
- la variabile *fieldCount%* contiene il numero di campi da cui è composto un record, ed è compresa tra 1 e *maxFields*;
- l'array *itemf$()* contiene le variabili di campo;
- l'array *itemLen%()* contiene la lunghezza di ciascun campo;
- l'array *itemName$()* contiene il nome di ciascun campo. Questi nomi non devono essere composti da più di *maxName* caratteri;
- la variabile *statusf$* rappresenta il primo campo di ogni record. Il carattere contenuto in questo campo può essere uno spazio o un asterisco. Un asterisco contrassegna il record corrente come record da cancellare;
- la variabile *recordLength%* contiene la lunghezza dei record del database. Il programma calcola la lunghezza sommando i valori di *itemLen%()* e aggiungendo 1 per comprendere anche *statusf$*;
- la variabile *lastRecord%* contiene il numero dell'ultimo record del database. Inizialmente, il programma calcola *lastRecord%* usando la formula LOF(1)/ *recordLength%*. Nelle altre situazioni, ad esempio quando si aggiunge un record, il valore di *lastRecord%* viene calcolato sommando 1 al valore precedente.

Il programma avrebbe dovuto usare queste variabili come parametri di procedura e passarle, quando necessario, alle varie procedure. Tuttavia, queste variabili sono state definite come globali e vengono condivise quando richiesto poiché appaiono nell'intero programma.

Il Programma 10.22 illustra molte delle tecniche presentate in questo libro. Se si esaminano le varie procedure, si potranno scoprire alcuni trucchi di programmazione che non sono stati trattati dettagliatamente nei capitoli precedenti.

Dopo essere partiti da semplici calcoli con le variabili intere, è stato creato un programma completo per la gestione di un database. Se sono stati seguiti attentamente

tutti gli argomenti proposti in questo libro, non solo si saprà utilizzare QBasic, ma si avrà una buona conoscenza delle moderne tecniche di programmazione.

Programma 10.22 *Un programma per la gestione di database*

```
REM Un programma di database generico         [10-22]
REDIM itemlen%(1)
REDIM itemf$(1), itemName$(1)
CONST true = -1, false = 0, maxName = 15, maxFields = 10
dataFile$ = ""
DO
  CALL DisplayMenu
  action$ = UCASE$(INPUT$(1))
  PRINT
  SELECT CASE action$
    CASE "1":  CALL AppendRecord
    CASE "2":  CALL CreateDatabase
    CASE "3":  CALL DisplayStructure
    CASE "4":  CALL GetRecordNumber("Numero di record", n%)
               CALL EditRecord(n%)
    CASE "5":  CALL DisplayDatabaseFiles
    CASE "6":  CALL InsertRecord
    CASE "7":  CALL ListRecords
    CASE "8":  CALL GetDatabaseName
               CALL OpenDatabase
    CASE "9":  CALL PackDatabase
    CASE "B": CALL RemoveOrRestoreRecord(n%)
    CASE "C": CALL SortDatabase
    CASE ELSE
  END SELECT
LOOP UNTIL action$ = "A"
CLS
IF dataFile$ <> "" THEN CALL CloseDatabase
PRINT "Database chiuso"
END

noFiles:
  PRINT "Non ci sono file di database in "; dd$
  RESUME NEXT

SUB AppendRecord
  SHARED fieldCount%, itemf$(), statusf$, lastRecord%
  IF DatabaseIsOpen THEN
    lastRecord% = lastRecord% + 1
    LSET statusf$ = " "
    FOR index% = 1 TO fieldCount%
      LSET itemf$(index%) = ""
    NEXT index%
    PUT #1, lastRecord%
    CALL EditRecord(lastRecord%)
```

```
  END IF
END SUB

SUB CloseDatabase
  SHARED dataFile$, itemf$(), itemlen%(), itemName$()
  CLOSE #1
  ERASE itemf$, itemlen%, itemName$
  dataFile$ = ""
END SUB

SUB CreateDatabase
  SHARED dataFile$
  DIM fieldName$(1 TO maxFields), fieldLen%(1 TO maxFields)
  'Se è aperto un altro database,lo chiude prima di procedere
  IF dataFile$ <> "" THEN CALL CloseDatabase
  CLS
  INPUT "Nome del database (massimo 8 lettere)"; dataFile$
  IF dataFile$ = "" THEN EXIT SUB
  dataFile$ = LEFT$(dataFile$, 8)
  count% = 0     'Conta il numero di campi specificati
  PRINT
  PRINT "Inserire un nome e una lunghezza di campo per riga."
  PRINT "Separare il nome e la lunghezza dei campi con una virgola."
  PRINT "Nome di campo limitato a"; maxName; "caratteri. "
  PRINT "Si può specificare un massimo di"; maxFields; "campi."
  PRINT
  PRINT "> ";
  LINE INPUT info$
  DO UNTIL (info$ = "") OR (count% = maxFields)
    comma = INSTR(info$, ",")
    IF comma <> 0 THEN
        count% = count% + 1
        fieldName$(count%) = LEFT$(LEFT$(info$, comma - 1), maxName)
        fieldLen%(count%) = VAL(RIGHT$(info$, LEN(info$) - comma))
        IF fieldLen%(count%) = 0 THEN count% = count% - 1
      ELSE
        PRINT "Errore di formato. Usare nome,lunghezza"
    END IF
    PRINT "> ";
    LINE INPUT info$
  LOOP
  IF count% > 0 THEN
      OPEN dataFile$ FOR OUTPUT AS #1
      WRITE #1, count%
      FOR index% = 1 TO count%
        WRITE #1, fieldName$(index%), fieldLen%(index%)
      NEXT index%
      CLOSE #1
      CALL OpenDatabase
    ELSE
```

```
      dataFile$ = ""
  END IF
END SUB

FUNCTION DatabaseIsOpen
  SHARED dataFile$
  IF dataFile$ = "" THEN
    PRINT "Non c'è un database aperto"
    CALL GetDatabaseName
    CALL OpenDatabase
  END IF
  IF dataFile$ = "" THEN
      DatabaseIsOpen = false
    ELSE
      DatabaseIsOpen = true
  END IF
END FUNCTION

SUB DisplayDatabaseFiles
  SHARED dd$
  CLS
  INPUT "Unità disco/Directory da visualizzare"; dd$
  ch$ = RIGHT$(dd$, 1)
  IF (ch$ <> ":") AND (ch$ <> "\") THEN dd$ = dd$ + "\"
  dd$ = dd$ + "*.DB"
  ON ERROR GOTO noFiles
  PRINT
  FILES dd$
  PRINT
  PRINT "Premere un tasto per continuare"
  a$ = INPUT$(1)
END SUB

SUB DisplayFieldTitles
  SHARED fieldCount%, itemlen%(), itemName$()
  CLS
  PRINT "*  Rec#";
  FOR index% = 1 TO fieldCount%
    il% = itemlen%(index%)
    PRINT " "; LEFT$(itemName$(index%) + SPACE$(il%), il%);
  NEXT index%
  PRINT
  PRINT
END SUB

SUB DisplayMenu
  CLS
  PRINT "<1> Aggiunge un record alla fine del database"
  PRINT "<2> Crea un database"
  PRINT "<3> Visualizza la struttura del database"
```

```
  PRINT "<4> Modifica il record specificato"
  PRINT "<5> Visualizza un elenco dei file di database"
  PRINT "<6> Inserisce un record prima di quello specificato"
  PRINT "<7> Visualizza i record"
  PRINT "<8> Apre un database esistente"
  PRINT "<9> Compatta il database"
  PRINT "<A> Uscita"
  PRINT "<B> Cancella o recupera un record"
  PRINT "<C> Ordina il database sulla base di un campo"
  PRINT
  PRINT "Comando?";
END SUB

SUB DisplayRecordForEdit (n%)
  SHARED fieldCount%, itemf$(), itemName$(), statusf$
  CLS
  PRINT USING "#####"; n%;
  IF statusf$ = "*" THEN PRINT "    REMOVED";
  PRINT
  FOR index% = 1 TO fieldCount%
    PRINT itemName$(index%); TAB(maxName + 1); ":"; itemf$(index%)
    PRINT
  NEXT index%
END SUB

SUB DisplayStructure
  SHARED fieldCount%, itemlen%(), itemName$()
  IF DatabaseIsOpen THEN
    CLS
    PRINT "Nome campo"; TAB(maxName + 2); "Lunghezza campo"
    FOR index% = 1 TO fieldCount%
      PRINT itemName$(index%); TAB(maxName + 2); itemlen%(index%)
    NEXT index%
    PRINT
    PRINT "Premere un tasto per continuare."
    a$ = INPUT$(1)
  END IF
END SUB

SUB EditRecord (n%)
  SHARED fieldCount%, itemf$()
  IF n% = 0 THEN EXIT SUB
  GET #1, n%
  CALL DisplayRecordForEdit(n%)
  FOR index% = 1 TO fieldCount%
    LOCATE 2 * index% + 1, maxName - 12, 1
    LINE INPUT "Nuovo valore> ", newItem$
    IF newItem$ <> "" THEN LSET itemf$(index%) = newItem$
  NEXT index%
  PUT #1, n%
```

```
END SUB

SUB GetDatabaseName
  SHARED dataFile$
  IF dataFile$ <> "" THEN CALL CloseDatabase
  PRINT
  INPUT "Database da aprire"; dataFile$
END SUB

SUB GetFieldNumber (n%)
  SHARED fieldCount%, itemName$()
  CLS
  PRINT "Numero  Nome"
  PRINT "campo   Campo"
  PRINT "------  ---------------"
  FOR index% = 1 TO fieldCount%
    PRINT USING " ##     &"; index%; itemName$(index%)
  NEXT index%
  PRINT
  DO
    INPUT "Numero di campo su cui basare l'ordinamento"; n%
  LOOP UNTIL (n% >= 0) AND (n% <= fieldCount%)
END SUB

SUB GetRecordNumber (message$, n%)
  SHARED lastRecord%
  n% = 0
  IF DatabaseIsOpen THEN
    PRINT
    DO
      PRINT message$;
      INPUT n%
    LOOP UNTIL (n% >= 0) AND (n% <= lastRecord%)
  END IF
END SUB

SUB InsertRecord
  SHARED fieldCount%, itemf$(), statusf$, lastRecord%
  IF DatabaseIsOpen THEN
    CALL GetRecordNumber("Numero di record", n%)
    IF n% <> 0 THEN
      FOR index% = lastRecord% TO n% STEP -1
        GET #1, index%
        PUT #1, index% + 1
      NEXT index%
      lastRecord% = lastRecord% + 1
      LSET statusf$ = " "
      FOR index% = 1 TO fieldCount%
        LSET itemf$(index%) = ""
      NEXT index%
```

```
      PUT #1, n%
      CALL EditRecord(n%)
    END IF
  END IF
END SUB

SUB ListRecords  '15 record alla volta
  SHARED fieldCount%, itemf$(), statusf$, lastRecord%
  IF DatabaseIsOpen THEN
    CALL DisplayFieldTitles
    FOR recNum% = 1 TO lastRecord%
      GET #1, recNum%
      PRINT USING "! #####"; statusf$; recNum%;
      FOR index% = 1 TO fieldCount%
        PRINT " "; itemf$(index%);
      NEXT index%
      PRINT
      IF (recNum% MOD 15 = 0) OR (recNum% = lastRecord%) THEN
        PRINT
        PRINT "Premere un tasto per continuare";
        a$ = INPUT$(1)
        IF recNum% <> lastRecord% THEN CALL DisplayFieldTitles
        PRINT
      END IF
    NEXT recNum%
  END IF
END SUB

SUB OpenAs2 (fileName$)
  SHARED recordLength%
  'si deve cancellare qualsiasi versione precedente di
  'fileName$ per assicurarsi che il file temporaneo aperto
  'sia vuoto. Tuttavia, la cancellazione di un file che non
  'esiste provoca un errore. Di conseguenza, prima di
  'cancellare fileName$, ne viene creato uno.
  OPEN fileName$ FOR OUTPUT AS 2
  CLOSE 2
  KILL fileName$
  OPEN fileName$ FOR RANDOM AS 2 LEN = recordLength%
END SUB

SUB OpenDatabase
  SHARED dataFile$, fieldCount%, itemf$(), itemlen%(), itemName$()
  SHARED recordLength%, lastRecord%
  IF dataFile$ <> "" THEN
    OPEN dataFile$ FOR INPUT AS #1
    INPUT #1, fieldCount%
    REDIM itemf$(1 TO fieldCount%)
    REDIM itemlen%(1 TO fieldCount%)
    REDIM itemName$(1 TO fieldCount%)
```

```
    FOR index% = 1 TO fieldCount%
      INPUT #1, itemName$(index%), itemlen%(index%)
    NEXT index%
    CLOSE #1
    recordLength% = 1
    FOR index% = 1 TO fieldCount%
      recordLength% = recordLength% + itemlen%(index%)
    NEXT index%
    OPEN dataFile$ + ".DB" FOR RANDOM AS #1 LEN = recordLength%
    CALL SetFields
    lastRecord% = LOF(1) / recordLength%
  END IF
END SUB

SUB PackDatabase
  SHARED dataFile$, statusf$, recordLength%, lastRecord%
  IF DatabaseIsOpen THEN
    PRINT
    PRINT "Packing " + dataFile$ + ".DB"
    FIELD #1, 1 AS statusf$, recordLength% - 1 AS file1f$
    CALL OpenAs2("zzzzzzzz.tmp")
    FIELD #2, recordLength% AS file2f$
    zindex% = 1
    FOR rec% = 1 TO lastRecord%
      GET #1, rec%
      IF statusf$ = " " THEN
        LSET file2f$ = statusf$ + file1f$
        PUT #2, zindex%
        zindex% = zindex% + 1
      END IF
    NEXT rec%
    holdName$ = dataFile$
    CLOSE #2
    CALL CloseDatabase
    KILL holdName$ + ".DB"
    NAME "zzzzzzzz.tmp" AS holdName$ + ".DB"
    dataFile$ = holdName$
    CALL OpenDatabase
  END IF
END SUB

SUB RemoveOrRestoreRecord (n%)
  SHARED statusf$
  message$ = "Numero di record da cancellare/recuperare"
  CALL GetRecordNumber(message$, n%)
  IF n% <> 0 THEN
    GET #1, n%
    IF statusf$ = "*" THEN
        LSET statusf$ = " "
        PRINT "Record recuperato"
```

```
      ELSE
        LSET statusf$ = "*"
        PRINT "Record cancellato"
    END IF
    PUT #1, n%
    'DELAY 1
  END IF
END SUB

SUB SetFields
  SHARED fieldCount%, itemf$(), itemlen%(), statusf$
  SELECT CASE fieldCount%
    CASE 1: FIELD #1, 1 AS statusf$, itemlen%(1) AS itemf$(1)
    CASE 2: FIELD #1, 1 AS statusf$, itemlen%(1) AS itemf$(1),_
           itemlen%(2) AS itemf$(2)
    CASE 3: FIELD #1, 1 AS statusf$, itemlen%(1) AS itemf$(1),_
           itemlen%(2) AS itemf$(2), itemlen%(3) AS itemf$(3)
    CASE 4: FIELD #1, 1 AS statusf$, itemlen%(1) AS itemf$(1),_
           itemlen%(2) AS itemf$(2), itemlen%(3) AS itemf$(3),_
           itemlen%(4) AS itemf$(4)
    CASE 5: FIELD #1, 1 AS statusf$, itemlen%(1) AS itemf$(1),_
           itemlen%(2) AS itemf$(2), itemlen%(3) AS itemf$(3),_
           itemlen%(4) AS itemf$(4), itemlen%(5) AS itemf$(5)
    CASE 6: FIELD #1, 1 AS statusf$, itemlen%(1) AS itemf$(1),_
           itemlen%(2) AS itemf$(2), itemlen%(3) AS itemf$(3),_
           itemlen%(4) AS itemf$(4), itemlen%(5) AS itemf$(5),_
           itemlen%(6) AS itemf$(6)
    CASE 7: FIELD #1, 1 AS statusf$, itemlen%(1) AS itemf$(1),_
           itemlen%(2) AS itemf$(2) itemlen%(3) AS itemf$(3),_
           itemlen%(4) AS itemf$(4), itemlen%(5) AS itemf$(5),_
           itemlen%(6) AS itemf$(6), itemlen%(7) AS itemf$(7)
    CASE 8: FIELD #1, 1 AS statusf$, itemlen%(1) AS itemf$(1),_
           itemlen%(2) AS itemf$(2), itemlen%(3) AS itemf$(3),_
           itemlen%(4) AS itemf$(4), itemlen%(5) AS itemf$(5),_
           itemlen%(6) AS itemf$(6), itemlen%(7) AS itemf$(7),_
           itemlen%(8) AS itemf$(8)
    CASE 9: FIELD #1, 1 AS statusf$, itemlen%(1) AS itemf$(1),_
           itemlen%(2) AS itemf$(2), itemlen%(3) AS itemf$(3),_
           itemlen%(4) AS itemf$(4), itemlen%(5) AS itemf$(5),_
           itemlen%(6) AS itemf$(6), itemlen%(7) AS itemf$(7),_
           itemlen%(8) AS itemf$(8), itemlen%(9) AS itemf$(9)
    CASE 10: FIELD #1, 1 AS statusf$, itemlen%(1) AS itemf$(1),_
           itemlen%(2) AS itemf$(2), itemlen%(3) AS itemf$(3),_
           iitemlen%(4) AS itemf$(4), itemlen%(5) AS itemf$(5),_
           itemlen%(6) AS itemf$(6), itemlen%(7) AS itemf$(7),_
           itemlen%(8) AS itemf$(8), itemlen%(9) AS itemf$(9),_
           itemlen%(10) AS itemf$(10)
  END SELECT
END SUB
```

```
SUB SortArrays (recNum%(), value$()) 'ordinamento bubble
  size% = UBOUND(recNum%, 1)
  FOR index1% = 1 TO size%
    exchanged% = false
    FOR index2% = 1 TO size% - index1%
      IF value$(index2%) > value$(index2% + 1) THEN
        'scambia index2% con index2%+1 in entrambi gli array
        temp$ = value$(index2%)
        value$(index2%) = value$(index2% + 1)
        value$(index2% + 1) = temp$
        temp% = recNum%(index2%)
        recNum%(index2%) = recNum%(index2% + 1)
        recNum%(index2% + 1) = temp%
        exchanged% = true
      END IF
    NEXT index2%
    IF NOT exchanged% THEN EXIT SUB
  NEXT index1%
END SUB

SUB SortDatabase
  SHARED dataFile$, itemf$(), itemName$()
  SHARED statusf$, recordLength%, lastRecord%
  IF DatabaseIsOpen THEN
    CALL GetFieldNumber(fieldNumber%)
    IF fieldNumber% = 0 THEN EXIT SUB
    PRINT
    PRINT "Ordinamento del file "; dataFile$;
    PRINT ".DB in base a "; itemName$(fieldNumber%)
    DIM recNum%(1 TO lastRecord%), value$(1 TO lastRecord%)
    FOR index% = 1 TO lastRecord%
      GET #1, index%
      recNum%(index%) = index%
      value$(index%) = itemf$(fieldNumber%)
    NEXT index%
    CALL SortArrays(recNum%(), value$())
    FIELD #1, 1 AS statusf$, recordLength% - 1 AS file1f$
    CALL OpenAs2("zzzzzzzz.tmp")
    FIELD #2, recordLength% AS file2f$
    FOR index% = 1 TO lastRecord%
      GET #1, recNum%(index%)
      LSET file2f$ = statusf$ + file1f$
      PUT #2, index%
    NEXT index%
    holdName$ = dataFile$
    CLOSE #2
    CALL CloseDatabase
    KILL holdName$ + ".DB"
    NAME "zzzzzzzz.tmp" AS holdName$ + ".DB"
    dataFile$ = holdName$
```

```
    CALL OpenDatabase
  END IF
END SUB
```

APPENDICE A

TABELLA ASCII

32	(spazio)	52	4	72	H	92	\
33	!	53	5	73	I	93	]
34	"	54	6	74	J	94	^
35	#	55	7	75	K	95	_
36	$	56	8	76	L	96	`
37	%	57	9	77	M	97	a
38	&	58	:	78	N	98	b
39	'	59	;	79	O	99	c
40	(	60	<	80	P	100	d
41	)	61	=	81	Q	101	e
42	*	62	>	82	R	102	f
43	+	63	?	83	S	103	g
44	,	64	@	84	T	104	h
45	-	65	A	85	U	105	i
46	.	66	B	86	V	106	j
47	/	67	C	87	W	107	k
48	0	68	D	88	X	108	l
49	1	69	E	89	Y	109	m
50	2	70	F	90	Z	110	n
51	3	71	G	91	[	111	o

112	p	148	ö	184	╕	220	▄
113	q	149	ò	185	╣	221	▌
114	r	150	û	186	║	222	▐
115	s	151	ù	187	╗	223	▀
116	t	152	ÿ	188	╝	224	∝
117	u	153	Ö	189	╜	225	β
118	v	154	Ü	190	╛	226	Γ
119	w	155	¢	191	┐	227	π
120	x	156	£	192	└	228	Σ
121	y	157	¥	193	┴	229	σ
122	z	158	₧	194	┬	230	µ
123	{	159	ƒ	195	├	231	τ
124	|	160	á	196	─	232	Φ
125	}	161	í	197	┼	233	θ
126	~	162	ó	198	╞	234	Ω
127		163	ú	199	╟	235	δ
128	Ç	164	ñ	200	╚	236	∞
129	ü	165	Ñ	201	╔	237	φ
130	é	166	ª	202	╩	238	∈
131	â	167	º	203	╦	239	∩
132	ä	168	¿	204	╠	240	≡
133	à	169	⌐	205	═	241	±
134	å	170	¬	206	╬	242	≥
135	ç	171	½	207	╧	243	≤
136	ê	172	¼	208	╨	244	⌠
137	ë	173	¡	209	╤	245	⌡
138	è	174	«	210	╥	246	÷
139	ï	175	»	211	╙	247	≈
140	î	176	░	212	╘	248	°
141	ì	177	▒	213	╒	249	∙
142	Ä	178	▓	214	╓	250	·
143	Å	179	│	215	╫	251	√
144	É	180	┤	216	╪	252	^{n}
145	æ	181	╡	217	┘	253	^{2}
146	Æ	182	╢	218	┌	254	■
147	ô	183	╖	219	█	255	

APPENDICE B

FUNZIONI INCORPORATE

Nota: *x* indica dei valori che contengono, normalmente, delle cifre decimali. I valori rappresentati con *n* e *m* sono numeri interi e, nel caso venga assegnata una parte frazionaria, vengono automaticamente arrotondati all'intero più vicino.

Funzioni relative agli array

LBOUND(*nomeArray*(*n*))	L'indice più piccolo utilizzabile nell'array.
UBOUND(*nomeArray*(*n*))	L'indice più grande utilizzabile nell'array.

Funzioni di conversione

CDBL(*x*)	Il numero a precisione doppia determinato da *x*.
CINT(*x*)	L'intero determinato da *x*.
CLNG(*x*)	L'intero lungo determinato da *x*.
CSNG(*x*)	Il numero a precisione singola determinato da *x*.
CVD(*a$*)	Il numero a precisione doppia contenuto in *a$*.
CVI(*a$*)	L'intero contenuto in *a$*.
CVL(*a$*)	L'intero lungo contenuto in *a$*.

CVS(*a$*)	Il numero a precisione singola contenuto in *a$*.
CVDMBF(*a$*)	Il numero a precisione doppia contenuto in *a$* nel BASIC standard o nelle versioni di QuickBASIC precedenti alla 4.0.
CVSMBF(*a$*)	Il numero a precisione singola contenuto in *a$* nel BASIC standard o nelle versioni di QuickBASIC precedenti alla 4.0.
HEX$(*n*)	La stringa contenente la rappresentazione esadecimale di *n*.
MKD$(*x*)	La stringa contenente il numero a precisione doppia *x*.
MKI$(*n*)	La stringa contenente l'intero *n*.
MKL$(*n*)	La stringa contenente il numero intero lungo *n*.
MKS$(*x*)	La stringa contenente il numero a precisione singola *x*.
MKDMBF$(*x*)	La stringa contenente il numero a precisione doppia *x* nel BASIC standard o nelle versioni di QuickBASIC precedenti alla 4.0.
MKSMBF$(*x*)	La stringa contenente il numero a precisione singola *x* nel BASIC standard o nelle versioni di QuickBASIC precedenti alla 4.0.
OCT$(*n*)	La stringa contenente la rappresentazione ottale di *n*.
STR$(*x*)	La stringa contenente la rappresentazione decimale di *x*.
VAL(*a$*)	Il valore numerico di *a$*, dove *a$* deve poter essere interpretato come numero.

Funzioni per il controllo dell'errore

ERDEV	Il numero di errore restituito dalla periferica che lo ha causato.
ERDEV$	Il nome della periferica che ha causato l'errore.
ERR	Il numero di errore verificatosi.
ERL	Il numero di riga in cui si è verificato l'errore.

Funzioni relative ai file

EOF(*n*)	-1 se è stata raggiunta la fine del file *n*, 0 in caso contrario.

LOC(*n*)	File sequenziale *n*: Il numero di blocchi da 128 caratteri letti o scritti. File ad accesso casuale *n*: Il numero del record corrente. File binario *n*: Il numero di byte dalla posizione corrente all'inizio del file.
LOF(*n*)	Il numero di caratteri nel file *n*.

Funzioni grafiche

PMAP(*x*, *n*)	Le coordinate fisiche (*n*=0) o naturali (*n*=2) di *x*.
PMAP(*y*, *n*)	Le coordinate fisiche (*n*=1) o naturali (*n*=3) di *y*.
POINT(*x*,*y*)	Il numero del colore associato al punto con coordinate (*x*,*y*).
POINT(*n*)	Le coordinate dell'ultimo punto indirizzato indicato da *n*.

Funzioni di inserimento

INKEY$	I(l) carattere(i) presente(i) nel buffer di tastiera.

Funzioni matematiche

ABS(*x*)	Il valore assoluto di *x*.
ATN(*x*)	L'arcotangente dell'angolo di *x* radianti.
COS(*x*)	Il coseno dell'angolo di *x* radianti.
EXP(*x*)	Il logaritmo naturale in base e (2,71828...) elevato alla potenza *x*: e^x.
FIX(*x*)	Il numero intero ottenuto scartando la parte decimale di *x*.
INT(*x*)	Il numero intero più grande minore o uguale a *x*.
LOG(*x*)	Il logaritmo naturale di *x*.
RND	Un numero pseudo-casuale compreso tra 0 e 1 (1 escluso).
SIN(*x*)	Il seno dell'angolo di *x* radianti
SGN(*x*)	Il segno di *x* (+1 se $x>0$, 0 se $x=0$ e -1 se $x<0$).
SQR(*x*)	La radice quadrata di *x*.
TAN(*x*)	La tangente dell'angolo di *x* radianti.

Funzioni di stampa

LPOS(1)	La 'posizione corrente del cursore' nel buffer di stampa.
SPC(*n*)	Stampa *n* spazi.
TAB(*n*)	Si posta sulla posizione *n* nella riga corrente.

Funzioni di schermo

CSRLIN	La riga dello schermo su cui è posizionato il cursore.
POS	Il numero di colonna su cui è posizionato il cursore.
SCREEN(*r,c*)	Il valore ASCII del carattere che si trova nella riga *r* e nella colonna *c*.
SCREEN(*r,c*,1)	Il colore del carattere che si trova nella riga *r* e nella colonna *c*.

Funzioni per la manipolazione delle stringhe

ASC(*a$*)	Il valore ASCII del primo carattere della stringa *a$*.
CHR$(*n*)	Il carattere il cui codice ASCII è *n*.
INSTR(*a$,b$*)	La posizione in cui *b$* è contenuta in *a$*.
INSTR(*n,a$,b$*)	La posizione a partire dal carattere *n* in cui *b$* è contenuta in *a$*.
LCASE$(*a$*)	La stringa *a$* convertita in lettere minuscole.
LEFT$(*a$,n*)	La stringa composta dai primi *n* caratteri di *a$*.
LEN(*a$*)	Il numero di caratteri contenuti in *a$*.
LTRIM$(*a$*)	La stringa *a$* senza gli spazi in testa.
MID$(*a$,m*)	La stringa composta dai caratteri di *a$* partendo dalla posizione *m*.
MID$(*a$,m,n*)	La stringa composta dagli *n* caratteri di *a$* partendo dalla posizione *m*.
RIGHT$(*a$,n*)	La stringa composta dagli ultimi *n* caratteri di *a$*.
RTRIM$(*a$*)	La stringa *a$* senza gli spazi in coda.
SPACE$(*n*)	Una stringa composta da *n* spazi.
STRING$(*n,a$*)	La stringa composta dal primo carattere di *a$* ripetuto *n* volte.

STRING$(*n*,*m*)	La stringa composta dal carattere con codice ASCII *m* ripetuto *n* volte.
UCASE$(*a$*)	La stringa *a$* convertita in lettere maiuscole.

Funzioni di sistema

ENVIRON$(*a$*)	La parte destra dell'equazione nella tabella di ambiente specificata da *a$*.
ENVIRON$(*n*)	L'equazione *n* nella tabella di ambiente.
FRE(" ")	La quantità di memoria disponibile per caricare nuovi dati.
FRE(-1)	La quantità di memoria disponibile per nuovi array numerici.
FRE(-2)	La dimensione più piccola dello stack verificatasi durante l'esecuzione del programma.
INP(*n*)	Il valore del byte letto dalla porta *n*.
IOCTL$(*n*)	La stringa di controllo restituita dopo aver attivato la periferica *n*.
PEEK(*n*)	Il valore del byte a distanza *n* nel segmento corrente.
SADD(*a$*)	La distanza nel segmento dati di default del primo byte di *a$*.
SETMEM(*n*)	La dimensione dell'heap dopo aver cambiato la sua dimensione di *n*.
VARPTR(*var*)	La distanza in VARSEG(*var*) del valore o del descrittore per *var*.
VARSEG(*var*)	Il segmento di memoria contenente il valore o il descrittore per *var*.
VARPTR$(*var*)	La stringa di cinque caratteri che identifica il tipo e la posizione di *var*.

Funzioni data e ora

DATE$	La data corrente nel formato "mm-gg-aaaa".
TIME$	L'ora corrente espressa come stringa nella forma "oo:mm:ss".
TIMER	Il numero di secondi passati da mezzanotte.

Funzioni varie

PEN(*n*)	Lo stato della penna luminosa indicata da *n*.
PLAY(0)	Il numero di note rimanenti nel buffer.
STICK(*n*)	Le coordinate del joystick indicato da *n*.
STRIG(*n*)	Lo stato del pulsante del joystick indicato da *n*.

APPENDICE C

CONVERSIONE IN QBASIC DEI PROGRAMMI IN BASIC STANDARD

COMANDI DA ELIMINARE

ENUNCIATI NON SUPPORTATI

I comandi del BASIC standard seguenti non sono supportati da QBasic:

```
AUTO RENUM
DELETE EDIT LIST LLIST LOAD MERGE BNEW SAVE
CONT
DEF USR   USR   MOTOR
```

I comandi nelle prime tre righe, conosciuti come comandi immediati, in genere non vengono eseguiti dall'interno dei programmi. Gli enunciati AUTO e RENUM agiscono sui numeri di riga e non sono supportati da QBasic. I comandi nella seconda riga, invece, trovano degli equivalenti nel menu File o nell'editor.

Quando si collauda un programma scritto in BASIC standard, lo si può interrompere in un punto qualsiasi, esaminare i valori delle variabili, e riprenderne l'esecuzione con

il comando CONT. In QBasic, invece, si può riprendere l'esecuzione di un programma precedentemente interrotto premendo il tasto F5.

I comandi CALL, DEF USR e USR, usati nel BASIC standard per accedere a un programma in linguaggio macchina, non sono supportati da QBasic. In QBasic, si utilizza il comando CALL per richiamare un sottoprogramma.

Alcuni computer (esclusi gli IBM PC, XT, AT o PS/2) dispongono di un connettore speciale a cui è possibile collegare un lettore di cassette. In questo caso, si può usare il comando MOTOR per attivare e disattivare il motore del lettore. Questa funzione viene usata raramente nei programmi in BASIC standard.

Gli enunciati seguenti, disponibili solo dalla versione 2.1 del BASIC standard per l'IBM PCjr, non sono supportati da QBASIC: PCOPY, NOISE, SOUND ON, SOUND OFF e TERM. Inoltre, in QBasic non sono disponibili le modalità SCREEN per la CGA numerate da 3 a 6, il parametro voice per il comando SOUND e il parametro 'video memory' per l'enunciato CLEAR.

VARIAZIONI NON SUPPORTATE DI ENUNCIATI SUPPORTATI

`CHAIN` *filespec*`,, ALL`	Elenca tutte le variabili in un enunciato COMMON
`CHAIN MERGE` *filespec*	
`CLEAR,` *n*	
`VARPTR(`*#filenum*`)`	

COMANDI CHE DEVONO ESSERE CONVERTITI

Nel BASIC standard, l'enunciato CALL viene usato solamente per richiamare una subroutine in linguaggio macchina. Un enunciato nella forma

```
CALL numvar (var1, var2, ...)
```

deve essere convertito nel comando QBasic

```
CALL ABSOLUTE (var1, var2, ..., numvar)
```

In BASIC standard, l'enunciato

```
CHAIN filespec
```

passa il controllo a un qualsiasi programma.

In BASIC standard, quando l'enunciato

```
COMMON var1, var2, ...
```

appare in un programma che ne richiama un altro tramite il comando CHAIN, le variabili specificate vengono passate al programma concatenato. In QBasic, l'enunciato COMMON deve essere presente anche nel programma che viene richiamato.

Nel BASIC standard, la funzione FRE restituisce sempre la quantità di memoria correntemente disponibile. In QBasic, invece, si possono ottenere diverse informazioni a seconda dell'argomento specificato.

ALTRE CONSIDERAZIONI

In QBasic ci sono molte parole riservate che possono invece essere usate come variabili nel BASIC standard. Tra queste, le più significative sono: ABSOLUTE, AS,BINARY, CASE, CONST, DECLARE, DO, DOUBLE, EXIT, FREEFILE, FUNCTION, LBOUND, LOCAL, LOOP, PALETTE, SEEK, SELECT, SHARED, SLEEP, SUB, UBOUND e UNTIL.

In BASIC standard, i numeri con parte frazionaria uguale a 0,5 vengono arrotondati per eccesso, a differenza di QBasic che li arrotonda all'intero pari più vicino. Si tenga presente che tutti gli enunciati di QBasic che utilizzano numeri interi procedono in questo modo. Ad esempio, il comando LOCATE 3.5,4.5 equivale a LOCATE 4.4.

Se, nel momento in cui termina l'esecuzione di un programma in BASIC standard, l'ultimo enunciato PRINT sopprime la coppia di caratteri CR/LF usando un punto e virgola o una virgola, il BASIC standard aggiunge automaticamente questa coppia di caratteri. Dato che QBasic non opera in questo modo, un programma può iniziare a visualizzare dei dati partendo dalla posizione in cui era arrivato il programma precedente. Per evitare questa situazione è sufficiente impartire un comando CLS.

In BASIC standard, tutti gli array vengono allocati in memoria dinamicamente. Quindi, quando vengono cancellati, si libera dello spazio in memoria. In QBasic, la memoria viene riservata in modo statico, quando possibile, per accelerare l'accesso agli array. Tuttavia, questo spazio di memoria non può essere recuperato fino al termine del programma. Se la quantità di memoria disponibile costituisce un problema, si può utilizzare il metacomando ¢DYNAMIC per creare degli array che possano essere rimossi completamente dalla memoria.

In BASIC standard, quando si esegue un programma, la modalità video non viene riportata alla sua impostazione di default, SCREEN 0,0. Ciò avviene invece in QBasic.

In BASIC standard, si possono salvare i programmi in tre formati: binario compresso, ASCII e protetto. In QBasic si possono eseguire solo i file in formato ASCII (quelli salvati con il comando SAVE *nomefile*,A). Se appaiono dei caratteri inusuali quando si carica in QBasic un programma scritto in BASIC standard, ci sono molte probabilità che il file sia stato salvato in formato binario o protetto. I programmi in formato binario possono essere convertiti in ASCII caricandoli in BASIC standard e salvandoli in formato ASCII; quelli protetti, invece, sono molto più difficili da convertire. Nella maggior parte delle versioni del BASIC standard, per convertire un file protetto in ASCII si può procedere nel modo seguente:

1. eseguire il programma seguente:

```
10 OPEN "CONVERTE.BAS" FOR OUTPUT AS #1
20 PRINT #1, CHR$(255);
30 CLOSE #1
```

2. caricare (LOAD) il programma protetto in memoria;
3. digitare LOAD "CONVERTE.BAS";
4. salvare (SAVE) il programma corrente in formato ASCII.

Si raccomanda di rimuovere tutti i numeri di riga non necessari per le routine di gestione degli errori e a cui non viene fatto riferimento. Questi non sono necessari in QBasic e occupano inutilmente dello spazio in memoria. Inoltre, rallentano il compilatore e potrebbero generare un messaggio di errore.

APPENDICE D

IL DEBUGGER DI QBASIC

Il debugger di QBasic è uno strumento molto utile per verificare e collaudare i programmi. Il debugger consente al programmatore di tracciare le istruzioni di un programma e di modificare ed esaminare i valori delle variabili durante l'esecuzione. Per richiamare il debugger di QBasic, si selezioni l'opzione Debug dalla barra dei menu.

MODALITÀ PASSO A PASSO

La caratteristica più importante del debugger, conosciuta come modalità passo a passo, è la possibilità di eseguire un'istruzione alla volta. Se non si potesse eseguire un comando alla volta, il programma giungerebbe rapidamente al termine e sarebbe molto difficile rilevare la presenza di eventuali errori.

La modalità passo a passo viene gestita tramite i tasti funzione F8 e F10. Quando si preme F8, il primo enunciato del programma viene evidenziato. Una seconda pressione del tasto F8 esegue il comando ed evidenzia quello immediatamente successivo. Ogni pressione successiva di F8 indica a QBasic di eseguire l'istruzione selezionata e di evidenziare quella seguente. La modalità passo a passo è il modo migliore per seguire il flusso di un programma attraverso le strutture di decisione, i

sottoprogrammi e le funzioni. Quando è evidenziata una chiamata a una procedura e si preme il tasto F8, la barra evidenziatrice si sposta sulla prima riga della procedura. Dopo l'esecuzione di tutti gli enunciati della procedura, la barra evidenziatrice ritorna sul comando seguente a quello che ha originato la chiamata. Si può visualizzare in qualsiasi momento lo schermo di output premendo il tasto F4.

In diverse situazioni, si potrebbe non voler eseguire una procedura in modalità passo a passo. In questo caso, è sufficiente premere il tasto F10 nel momento in cui viene evidenziata la chiamata alla procedura. Con questo tasto, la procedura viene eseguita senza interruzione e la barra evidenziatrice si posiziona sul comando successivo a quello che ha originato la chiamata. Ci si ricordi che F8 'traccia' una procedura, mentre F10 la esegue normalmente.

VISUALIZZAZIONE DEL VALORE DI UNA VARIABILE

Durante l'esecuzione di un programma in modalità passo a passo, può risultare utile sapere il valore di una determinata variabile in un certo punto del programma. A questo scopo, ci si sposti nella finestra Immediato premendo il tasto F6, si digiti CLS:PRINT seguito dal nome della variabile e si prema Invio. Ad esempio, per visualizzare il valore di *miaVariabile*, si digiti CLS:PRINT miaVariabile e si prema Invio. Verrà mostrato lo schermo di Output e il valore della variabile sarà visualizzato. A questo punto, si prema un tasto qualsiasi per ritornare all'ambiente di QBasic e quindi F6 per riposizionarsi sulla riga corrente del programma.

Nota: Se si visualizza il valore di una variabile in questo modo, si cancella il contenuto corrente dello schermo di output.

MODIFICA DI UN VALORE

Durante il collaudo di un programma, potrebbe essere utile modificare il valore di una variabile mentre l'esecuzione è sospesa. Ad esempio, si potrebbe cambiare il valore di una variabile per uscire da un ciclo nel caso in cui non si verifichi mai la condizione di uscita. QBasic consente di modificare il valore di una variabile in un modo molto semplice. Quando il programma è in attesa, si prema F6 per spostare il cursore nella finestra Immediato, e si digiti il nome della variabile da modificare seguito dal segno uguale e dal nuovo valore da assegnare. Ad esempio, per assegnare a *r* il valore 3, si digiti r=3. Dopo aver premuto Invio, la variabile avrà il nuovo valore specificato. Si prema a questo punto il tasto F6 per riportare il cursore nella finestra di visualizzazione.

PUNTI DI INTERRUZIONE

Un punto di interruzione consente di specificare una riga in cui sospendere l'esecuzione del programma. Una volta interrotto il programma, si possono visualizzare e modificare le variabili seguendo le procedure esposte nelle sezioni precedenti. Dopo aver interrotto un programma, si può riprendere l'esecuzione dall'istruzione successiva al punto di interruzione premendo il tasto F5, o dall'inizio del programma premendo la combinazione di tasti Maiusc-F5.

Per definire un punto di interruzione, si sposti il cursore sulla riga desiderata e si prema F9. La riga selezionata viene evidenziata. A questo punto, dopo aver avviato il programma, l'esecuzione verrà sospesa nel momento in cui viene raggiunta la riga specificata. Si possono impostare più punti di interruzione in un programma. Ogni volta che viene incontrato un punto di interruzione, l'esecuzione del programma viene sospesa. Per rimuovere un punto di interruzione, ci si sposti sulla riga che lo contiene e si prema F9.

UN ESERCIZIO CON IL DEBUGGER

L'esercizio seguente mette in pratica le nozioni finora esposte.

1. Si digiti il programma seguente:

```
CLS
INPUT "Costo orario ", costo
IF costo < 3800 THEN
   PRINT "Inferiore al valore minimo"
ELSE
   PRINT "Ok"
   PRINT "Il reddito annuo è di circa ";
   PRINT USING "##,###"; Reddito(costo)
END IF

FUNCTION Reddito (stipendio)
   Reddito = 2000000*stipendio
END FUNCTION
```

2. Ritornare al programma principale usando i tasti Maiusc-F2. Spostare il cursore sull'enunciato PRINT “Ok” e premere F9. Si noti che la riga viene evidenziata. La riga è stata definita come punto di interruzione;
3. eseguire il programma premendo i tasti Maiusc-F5 e digitare 6500 in risposta alla richiesta “Costo orario”;
4. premere F4 per visualizzare lo schermo di output. Si noti che la riga PRINT “Ok” non è stata eseguita. Premere F4 per ritornare alla finestra di visualizzazione;

5. premere F5 per continuare l'esecuzione dal punto di interruzione fino al termine del programma (dopo aver visualizzato lo schermo di output, si prema un tasto qualsiasi per tornare alla finestra di visualizzazione);
6. premere Maiusc-F5 per eseguire nuovamente il programma dall'inizio e digitare, questa volta, 2750. Si noti che il programma viene eseguito fino in fondo senza interruzioni. Si prema un tasto qualsiasi per tornare alla finestra di visualizzazione;
7. rimuovere il punto di interruzione precedentemente impostato spostando il cursore sulla riga PRINT "Ok" e premendo F9. La riga non è più evidenziata e il punto di interruzione viene rimosso;
8. premere F8 alcune volte per eseguire il programma. Premere un tasto qualsiasi per tornare alla finestra di visualizzazione;
9. premere F10 alcune volte per eseguire il programma. Si noti che la funzione viene eseguita come un singolo enunciato;
10. premere Maiusc-F2 per visualizzare la funzione Reddito. Spostare il cursore sulla riga Reddito=2000000*stipendio e premere F9 per definire un punto di interruzione;
11. premere Maiusc-F5 per eseguire il programma e digitare 5000 come costo orario;
12. quando l'esecuzione del programma viene sospesa, modificare il valore di *stipendio* premendo F6 e digitando stipendio=3250. Premere F6 per tornare alla finestra di visualizzazione;
13. premere F5 per continuare l'esecuzione del programma. Si noti che la finestra di dialogo contiene la parola Ok; ciò accade perché il valore inizialmente inserito, 5000, era superiore al minimo consentito. Tuttavia, il calcolo del reddito annuo viene effettuato considerando un costo orario di 3250. Si prema un tasto qualsiasi per tornare alla finestra di visualizzazione.

APPENDICE E

L'AMBIENTE DI QBASIC

Questa appendice illustra tutti i comandi presenti nei menu di QBasic.

I COMANDI DEL MENU FILE

Se si preme F dalla barra dei menu, o Alt-F dalla finestra Immediato o di visualizzazione, viene richiamato il menu a tendina File che contiene sei comandi.

Nuovo
Il comando Nuovo cancella dalla memoria il programma corrente e consente l'inserimento di un nuovo programma. Alla nuova finestra viene assegnato il nome Senza titolo. Se il comando Nuovo viene impartito prima di salvare eventuali modifiche apportate al programma corrente, QBasic chiede conferma dell'operazione.

Apri
Il comando Apri carica un programma memorizzato su disco nella finestra di visualizzazione, in modo da consentirne la modifica o l'esecuzione. Se viene impartito il comando Nuovo prima di aver salvato le modifiche apportate a un programma

caricato con Apri, QBasic richiede se si desidera salvare il file prima di cancellare il contenuto della memoria.

Il comando Apri richiama una finestra di dialogo per richiedere il file da caricare. Per caricare un programma, si digiti il nome del file nell'apposita casella (includendo, se necessario, il percorso completo) e si prema Invio. Se al nome del file è associata l'estensione standard BAS, non è necessario specificare l'estensione. Per specificare un file senza estensione, si deve aggiungere un punto alla fine del nome. Se il file richiesto non esiste, QBasic presume che si voglia creare un nuovo programma da salvare successivamente con il nome specificato.

Un metodo alternativo per caricare un programma in memoria consiste nel selezionare il nome del file dalla casella di riepilogo che mostra tutti i file, con estensione .BAS, presenti nel disco, e premere Invio. Per spostare il cursore in questa casella si deve premere il tasto Tab.

L'unità disco e/o la directory dei programmi visualizzati nella casella di riepilogo dei file, può essere cambiata digitando il percorso nella casella Nome file e premendo Invio. Si possono anche utilizzare i metacaratteri, il punto di domanda e l'asterisco, per richiedere determinati gruppi di file. Il punto di domanda indica un qualsiasi singolo carattere, mentre un asterisco rappresenta una qualsiasi serie di caratteri. Ad esempio, se nella casella di riepilogo si digita A:\ANDREA\?R*.BAS, vengono visualizzati tutti i file presenti nell'unità disco A con estensione .BAS e il cui nome ha, come seconda lettera, la R.

Salva
Il comando Salva copia il programma correntemente in memoria su disco. Se al programma non è stato ancora assegnato un nome (cioè se alla finestra è associato il nome Senza titolo), QBasic richiama una finestra di dialogo per richiedere il nome da assegnare al file. Se il nome del file termina con un punto, non viene associata nessuna estensione. In caso contrario, a meno che non venga specificato diversamente, viene utilizzata automaticamente l'estensione di default .BAS. Il file viene salvato nella directory corrente dell'unità disco corrente, a meno che non venga inserito un percorso specifico.

Salva con nome
Il comando Salva con nome salva su disco il programma correntemente in memoria fornendo l'opportunità di memorizzare il file con un nome diverso da quello già assegnato. La finestra di dialogo è identica a quella richiamata dal comando Salva.

Stampa
Il comando Stampa invia alla stampante il testo correntemente selezionato, il contenuto della finestra o l'intero programma, a seconda dell'opzione selezionata.

Esci
Il comando Esci consente di uscire da QBasic e ritornare al DOS. Se il comando Esci viene impartito prima di salvare eventuali modifiche apportate al programma corrente, QBasic chiede conferma dell'operazione.

COMANDI DEL MENU MODIFICA

Se si preme E dalla barra dei menu, o Alt-E dalla finestra Immediato o di visualizzazione, viene richiamato il menu a tendina Modifica che contiene quattro comandi.

Taglia
Il comando Taglia rimuove il blocco di testo selezionato dalla finestra di visualizzazione e lo inserisce in Appunti.

Copia
Il comando Copia inserisce in Appunti il blocco di testo selezionato nella finestra di visualizzazione.

Incolla
Il comando Incolla inserisce il contenuto di Appunti nella finestra di visualizzazione, a partire dalla posizione del cursore.

Cancella
Il comando Cancella rimuove il testo selezionato senza salvarlo in Appunti (il contenuto di Appunti resta inalterato).

COMANDI DEL MENU VISUALIZZA

Se si preme V dalla barra dei menu, o Alt-V dalla finestra Immediato o di visualizzazione, viene richiamato il menu a tendina Visualizza che contiene tre comandi.

SUBs
Il comando SUBs visualizza una finestra di dialogo che consente di selezionare una procedura da visualizzare o cancellare.

Dividi

Il comando Dividi divide la finestra di visualizzazione orizzontalmente in modo da consentire l'accesso a due parti differenti del programma. I tasti F6 e Maiusc-F6 consentono di spostarsi tra le finestre. La dimensione della finestra attiva può essere aumentata tramite la combinazione di tasti Alt-Più, e diminuita mediante i tasti Alt-Meno. La combinazione di tasti Ctrl-F10 consente di ingrandire la finestra a pieno schermo e di riportarla alla sua dimensione originale.

Schermo output

Il comando Schermo output consente di attivare e disattivare la finestra di output. La stessa funzione viene svolta dal tasto F4.

COMANDI DEL MENU CERCA

Se si preme C dalla barra dei menu, o Alt-C dalla finestra Immediato o di visualizzazione, viene richiamato il menu a tendina Cerca che contiene tre comandi.

Trova

Il comando Trova cerca la prima occorrenza del testo specificato partendo dalla posizione del cursore.

Ripeti trova

Il comando Ripeti trova cerca l'occorrenza successiva del testo specificato in precedenza con il comando Trova.

Cambia

Il comando Cambia cerca la prima occorrenza del testo specificato partendo dalla posizione del cursore e lo sostituisce con il nuovo testo.

COMANDI DEL MENU ESEGUI

Se si preme E dalla barra dei menu, o Alt-E dalla finestra Immediato o di visualizzazione, viene richiamato il menu a tendina Esegui che contiene tre comandi.

Avvia
Il comando Avvia esegue il programma correntemente in memoria, iniziando dalla prima riga. Si può svolgere la stessa operazione premendo la combinazione di tasti Maiusc-F5.

Riavvia

Quando un programma viene sospeso prima del suo completamento, il comando Riavvia consente di riprendere l'esecuzione del programma dall'inizio.

Continua
Quando un programma viene sospeso prima del suo completamento, il comando Continua permette di riprendere l'esecuzione dal punto di interruzione.

COMANDI DEL MENU DEBUG

Se si preme D dalla barra dei menu, o Alt-D dalla finestra Immediato o di visualizzazione, viene richiamato il menu a tendina Debug che contiene sei comandi.

Passo
Il comando Passo consente di eseguire un programma un'istruzione alla volta. Ad ogni esecuzione del comando, viene impartita un'istruzione del programma.

Procedura passo
Il comando Procedura passo esegue il programma un'istruzione alla volta, senza utilizzare questa modalità durante le procedure. Le procedure vengono sempre eseguite come si fossero un singolo enunciato.

Analizza il flusso
Il comando Analizza il flusso consente di esaminare il flusso di istruzioni di un programma. Quando è abilitato, ogni enunciato viene evidenziato mentre viene eseguito.

Punto di interruzione
Un punto di interruzione è una riga di programma in cui l'esecuzione viene automaticamente sospesa. Il comando Punto di interruzione attiva e disattiva il punto di interruzione nella riga su cui è posizionato il cursore.

Elimina ogni punto di interruzione

Il comando Elimina ogni punto di interruzione rimuove tutti i punti di interruzione impostati nel programma.

Imposta istruzione successiva

Il comando Imposta istruzione successiva indica a QBasic che la riga su cui è posizionato il cursore è quella che dovrà essere eseguita successivamente.

COMANDI DEL MENU OPZIONI

Se si preme O dalla barra dei menu, o Alt-O dalla finestra Immediato o di visualizzazione, viene richiamato il menu a tendina Opzioni che contiene tre comandi.

Schermo

Il comando Schermo richiama una finestra di dialogo che può essere usata per personalizzare il colore di primo piano e di sfondo del testo digitato nella finestra di visualizzazione e degli enunciati evidenziati durante il collaudo, per selezionare e deselezionare le barre di scorrimento usate con il mouse, e per impostare le posizioni dei tabulatori.

Percorso di guida

Il comando Percorso di guida viene usato per indicare a QBasic la posizione del file QBASIC.HLP, che contiene le informazioni del sistema di aiuto. Normalmente, questo file si trova nella stessa directory in cui è stato installato QBASIC.EXE. Se così non fosse, si dovrebbe specificare il nuovo percorso per poter richiamare il sistema di aiuto. Un percorso completo consiste dell'unità disco seguita da una sequenza di nomi di sottodirectory.

Verifica sintassi

Normalmente, quando si inserisce una riga di testo nella finestra di visualizzazione, QBasic verifica automaticamente il contenuto della riga, converte in lettere maiuscole le parole chiave, e aggiunge degli spazi per migliorare la leggibilità. Questa caratteristica viene abilitata e disabilitata tramite il comando Verifica sintassi.

COMANDI DEL MENU ?

Se si preme ? dalla barra dei menu, o Alt-? dalla finestra Immediato o di visualizzazione, viene richiamato il menu a tendina ? che contiene cinque comandi.

Indice

Il comando Indice visualizza un elenco alfabetico delle parole chiave. Per ottenere delle informazioni su un determinato argomento, si posizioni il cursore sulla voce desiderata e si prema Invio o F1. Se si preme una lettera, il cursore si sposta sulla prima parola chiave che inizia con quella lettera.

Sommario

Il comando Sommario visualizza un menu contenente gli argomenti su cui sono disponibili delle informazioni. Dopo aver premuto Tab o un tasto di direzione per evidenziare l'argomento desiderato, si prema Invio o F1 per ottenere le informazioni del caso.

Argomento:

Quando il cursore è posizionato su una parola chiave nella finestra di visualizzazione, si può impartire il comando Argomento per ottenere delle informazioni su quella parola chiave.

Uso della Guida

Il comando Uso della Guida visualizza delle informazioni sul sistema di aiuto.

Informazioni su...

Il comando Informazioni su... riporta il numero e la data di copyright della versione di QBasic correntemente in uso.

APPENDICE F

PAROLE RISERVATE

Le parole seguenti non devono essere utilizzate come etichette o come nomi di variabili e di procedure.

ABS
ABSOLUTE
ACCESS
ALIAS
AND
ANY
APPEND
AS
ASC
ATN
BASE
BEEP
BINARY
BLOAD
BSAVE
BYVAL
CALL
CALLS
CASE
CDBL
CHAIN
CHDIR
CHR$
CINT
CIRCLE
CLEAR
CLNG
CLOSE
CLS
COLOR
COM
COMMAND$
COMMON
CONST
COS
CSNG

CSRLIN
CVD
CVDMBF
CVI
CVL
CVS
CVSMBF
DATA
DATE$
DECLARE
DEF
DEFDBL
DEFINT
DEFLNG
DEFSNG
DEFSTR
DIM
DO
DOUBLE
DRAW
ELSE
ELSEIF
END
ENDIF
ENVIRON
ENVIRON$
EOF
EQV
ERASE
ERDEV
ERDEV$
ERL
ERR
ERROR
EVENT
EXIT
EXP
FIELD
FILEATTR
FILES
FIX
FN
FOR
FRE
FREEFILE
FUNCTION
GET
GOSUB
GOTO
HEX$
IF
IMP
INKEY$
INP
INPUT
INPUT$
INSTR
INT
INTEGER
IOCTL
IOCTL$
IS
KEY
KILL
LBOUND
LCASE$
LEFT$
LEN
LET
LINE
LIST
LOC
LOCATE
LOCK
LOF
LOG
LONG
LOOP
LPOS
LPRINT
LSET
LTRIM$
MID$
MKD$
MKDIR
MKDMBF$
MKI$
MKL$
MKS$
MKSMBF$
MOD
NAME
NEXT

NOT
OCT$
OFF
ON
OPEN
OPTION
OR
OUT
OUTPUT
PAINT
PALETTE
PCOPY
PEEK
PEN
PLAY
PMAP
POINT
POKE
POS
PRESET
PRINT
PSET
PUT
RANDOM
RANDOMIZE
READ
REDIM
REM
RESET
RESTORE
RESUME
RETURN
RIGHT$
RMDIR
RND
RSET
RTRIM$
RUN
SADD
SCREEN
SEEK
SEG
SELECT
SETMEM
SGN
SHARED
SHELL
SIN
SINGLE
SLEEP
SOUND
SPACE$
SPC
SQR
STATIC
STEP
STICK
STOP
STR$
STRIG
STRING
STRING$
SUB
SWAP
SYSTEM
TAB
TAN
THEN
TIME$
TIMER
TO
TROFF
TRON
TYPE
UBOUND
UCASE$
UEVENT
UNLOCK
UNTIL
USING
VAL
VARPTR
VARPTR$
VARSEG
VIEW
WAIT
WEND
WHILE
WIDTH
WINDOW
WRITE
XOR

APPENDICE G

COMANDI, FUNZIONI E METACOMANDI DI QBASIC

Gli apici inseriti in alcune delle sezioni seguenti fanno riferimento all'elenco di argomenti riportati alla fine dell'appendice.

ABS La funzione ABS restituisce il valore assoluto di un numero. Se x è uguale a -3, ABS(x) restituisce il valore 3.

ASC Restituisce il codice ASCII del carattere specificato. La funzione ASC(*a$*) fornisce il codice ASCII del primo carattere della stringa *a$*.

ATN La funzione trigonometrica ATN, o arcotangente, è l'inverso della funzione tangente. Per un qualsiasi numero x, ATN(x) restituisce un angolo in radianti compreso tra -pi/2 e pi/2 la cui tangente è uguale a x[2].

BEEP L'enunciato BEEP genera un suono di frequenza pari a 800 Hz che dura una frazione di secondo.

BLOAD Il comando BLOAD *specFile,m* carica in memoria i byte contenuti nel file *specFile*, partendo dalla locazione *m* nel segmento corrente. Se il contenuto di determinate locazioni di memoria viene salvato con il comando BSAVE, è possibile recuperarlo in un secondo tempo tramite l'enunciato BLOAD *specFile*. Questa procedura viene spesso utilizzata per salvare e recuperare il contenuto dello schermo.[9, 4]

BSAVE Il comando BSAVE *specFile,n,m* salva nel file *specFile* il contenuto di *m* locazioni di memoria, partendo dalla locazione *n* del segmento corrente.[9, 4]

CALL Un enunciato nella forma CALL *NomeSottoprogramma*(*ListaArg*) viene usato per eseguire il sottoprogramma specificato, passando i valori e le variabili contenute in *ListaArg*. Per specificare degli array in questa lista, si deve inserire il nome seguito da una coppia di parentesi vuote. Il valore di una variabile passata come argomento può essere modificato dal sottoprogramma, a meno che questa non sia racchiusa tra parentesi. Una volta eseguiti tutti i comandi del sottoprogramma, il controllo ritorna all'istruzione successiva all'enunciato CALL.

Nota: La parola chiave CALL può essere omessa. In questo caso, devono essere omesse anche le parentesi; l'enunciato prende quindi la forma di *NomeSottoprogramma ListaArg*.

CALL ABSOLUTE Il comando CALL ABSOLUTE passa il controllo a una routine in linguaggio macchina, analogamente al modo in cui CALL richiama una procedura. L'enunciato CALL ABSOLUTE(*ListaArg, offvar*) richiama un programma in linguaggio macchina che inizia alla locazione *offvar* del segmento di memoria corrente. Gli argomenti vengono usati dal sottoprogramma in linguaggio macchina. Il comando CALLS ABSOLUTE funzione come CALL ABSOLUTE ad eccezione del fatto che deve essere passato sia il segmento che la distanza per ciascun argomento.[9]

CDBL La funzione CDBL converte dei numeri interi, interi lunghi e a precisione singola in valori a precisione doppia. Se *x* è un numero, il valore restituito da CDBL(*x*) è il numero a precisione doppia determinato da *x*.

CHAIN Il comando CHAIN passa il controllo dal programma corrente a un altro programma presente sul disco. L'enunciato CHAIN *specFile* carica ed esegue il codice sorgente del programma contenuto nel file *specFile* (aggiungendo l'estensione BAS se non ne viene specificata nessuna). Se si utilizza il comando COMMON in entrambi i programmi, è possibile passare i valori delle variabili del primo programma in quelle corrispondenti nel secondo programma.

CHDIR Il comando CHDIR *percorso* cambia la directory corrente nell'unità disco specificata con la sottodirectory indicata da *percorso*. Ad esempio, CHDIR "C:\" attiva la directory radice dell'unità C. Se la lettera dell'unità viene omessa, viene utilizzata l'unità disco di default.[3]

CHR$ Se *n* è un numero compreso tra 0 e 255, la funzione CHR$(*n*) restituisce il carattere il cui codice ASCII è uguale a *n*.

CINT La funzione CINT converte dei numeri interi lunghi, a precisione singola e a precisione doppia, in numeri interi. Se *x* è un numero compreso tra -32768 e 32767, il valore restituito da CINT(*x*) è il numero intero (eventualmente arrotondato) determinato da *x*.

CIRCLE L'enunciato CIRCLE(*x,y*)*,r,c,r1,r2,a* traccia un'ellisse (o solo una parte). Il centro dell'ellisse viene determinato dal punto (*x,y*) e il raggio più lungo è *r*. Il colore dell'ellisse viene determinato da *c*. Se vengono specificati *r1* e *r2*, QBasic traccia, in senso antiorario, solo la porzione dell'ellisse che si estende dal raggio con angolo di ABS(*r1*) radianti rispetto al raggio orizzontale al raggio con angolo di ABS(*r2*) radianti sempre rispetto al raggio orizzontale. Se *r1* o *r2* è negativo, viene tracciato anche il raggio. Il rapporto tra la lunghezza del diametro verticale e la lunghezza del diametro orizzontale viene determinato da *a*. Se *a* non viene specificato, QBasic traccia un cerchio.[2, 5, 6]

CLEAR Il comando CLEAR azzera tutte le variabili e gli elementi degli array statici, chiude tutti i file, rimuove dalla memoria tutti gli array dinamici, e reinizializza lo stack. Inoltre, se ,,*s* viene aggiunto dopo l'enunciato CLEAR, la dimensione dello stack viene impostata a *s*.[17, 1]

CLNG La funzione CLNG converte dei numeri interi, a precisione singola e a precisione doppia, in numeri interi lunghi.. Se *x* è un numero compreso tra -2.147.483.648 e 2.147.483.467, il valore restituito da CDBL(*x*) è il numero a intero lungo (eventualmente arrotondato) determinato da *x*.

CLOSE Il comando CLOSE #*n* chiude il file aperto in precedenza con il numero di riferimento *n*. Se usato da solo, CLOSE chiude tutti i file correntemente aperti.

CLS L'enunciato CLS cancella il contenuto dello schermo e posiziona il cursore in alto a sinistra dello schermo. Se è attiva una viewport grafica (si veda VIEW), il comando CLS cancella la viewport. L'enunciato CLS 0 cancella l'intero schermo, mentre CLS 1 cancella la viewport grafica attiva, se esistente; in caso contrario, cancella l'intero schermo. Il comando CLS 2 rimuove solo la viewport di testo (si veda VIEW PRINT).

COLOR In modalità testo (SCREEN 0), il comando COLOR produce degli effetti speciali (ad esempio, del testo sottolineato) o dei colori a seconda del monitor a cui è collegato il computer. L'enunciato COLOR *f,b,bd* imposta il colore di primo piano su *f*, il colore di sfondo su *b*, e il colore del bordo su *bd*, dove *f* può essere compreso tra 0 e 15 e *b* tra 0 e 7. Il comando COLOR *f*+16,*b,bd* seleziona gli stessi colori dell'enunciato precedente, impostando però un primo piano lampeggiante.

In modalità SCREEN 1, sono disponibili due palette di colori. Il comando COLOR *b,p* specifica *b* come colore di sfondo e *p* come palette da utilizzare. Il testo viene visualizzato con il colore 3 della palette selezionata e i grafici possono essere rappresentati con un qualsiasi colore di quella palette.

Nelle modalità EGA e VGA (7, 8 e 9) è disponibile una palette di 16 colori sia per il testo che per la grafica. L'enunciato COLOR *f,b* utilizza come colore di primo piano il colore numero *f*, e come colore di sfondo il colore numero *b*, dove *f* e *b* possono essere compresi tra 0 e 15.

Nelle modalità VGA e MCGA (12 e 13), il comando COLOR *f* utilizza come colore di primo piano il colore numero *f*. Il colore di sfondo viene impostato con il comando PALETTE 9,*c*.[5, 6]

COM(*n*) Il comando COM(*n*) attiva o disattiva la porta di comunicazione *n*, a seconda della parola chiave specificata (ON e OFF).[15, 10]

COMMON Se un enunciato nella forma COMMON *daVar1, daVar2,...,daVarN* precede un comando CHAIN, e un enunciato nella forma COMMON *aVar1, aVar2,...,aVarN* appare nel programma richiamato da CHAIN, il valore di *daVar1* viene assegnato alla variabile *aVar1*, il valore di *daVar2* alla variabile *aVar2*, e così via. Anche se i nomi delle variabili corrispondenti definite dal comando COMMON possono avere un nome diverso, è necessario che siano dello stesso tipo: stringa, intero, intero lungo, a precisione singola, a precisione doppia o record definiti dall'utente. Il tipo di ogni variabile viene determinato da un simbolo di dichiarazione. Quando si inserisce in un programma l'enunciato COMMON SHARED *var1, var2,...,varN*, le variabili specificate vengono condivise da tutte le procedure del programma. Il comando COMMON deve apparire prima di qualsiasi enunciato eseguibile e non può essere inserito in una procedura.

CONST L'enunciato CONST *nomeCostante=espressione* indica a QBasic di sostituire qualsiasi occorrenza di *nomeCostante* con il valore dell'*espressione*. La sostituzione ha luogo prima dell'esecuzione di una qualsiasi riga di programma. A differenza di LET, CONST non imposta una locazione di memoria per una variabile. Un determinato nome di costante può apparire in un solo enunciato CONST; *nomeCostante* viene chiamato *costante simbolica* o *costante con nome*.

COS La funzione trigonometrica COS(*x*) restituisce il coseno di un angolo di *x* radianti.[2]

CSNG La funzione CSNG converte dei numeri interi, interi lunghi o a precisione doppia, in numeri a precisione singola. Se *x* è un numero, il valore di CSNG(*x*) è il numero a precisione singola determinato da *x*.

CSRLIN La funzione CSRLIN restituisce, in qualsiasi momento, un numero che indica la riga dello schermo su cui è posizionato il cursore.[8]

CVI, CVL, CVS, CVD Quando si utilizza il metodo del buffer con i file ad accesso casuale, i numeri da memorizzare nei file tramite i comandi LSET e PUT devono prima essere convertiti in stringhe. Per riportare questi numeri al loro stato originale, si possono usare queste quattro funzioni. Se un intero è stato convertito nella stringa *a$* di lunghezza 2, il valore di CVI(*a$*) restituisce il valore originale. Analogamente, CVL(*a$*), CVS(*a$*) e CVD(*a$*) restituiscono, rispettivamente, il valore intero lungo, a precisione singola e a precisione doppia precedentemente convertito nella stringa *a$* di lunghezza 4, 4 e 8.[11]

CVSMBF, CVDMBF I file ad accesso casuale creati in BASIC, GW-BASIC, BASICA o nelle versioni precedenti di QuickBASIC, utilizzano un metodo diverso da quello di QBasic per memorizzare i numeri come stringhe. I numeri a singola e doppia precisione che sono stati convertiti in stringa e memorizzati in file ad accesso casuale da una di queste versione del BASIC, possono essere importati in QBasic usando queste due funzioni.[11]

DATA L'enunciato DATA *cost1, cost2,...,costN* contiene delle costanti che possono essere prelevate a assegnate a delle variabili tramite il comando READ.

**DATE$** La funzione DATE$ restituisce la data corrente sotto forma di stringa nella forma mm-gg-aaaa. Se *d$* è una stringa in questo formato, l'enunciato DATE$=*d$* imposta la data sul valore specificato da *d$*.

DECLARE L'enunciato opzionale DECLARE SUB *NomeSottoprogramma (par1,par2,...)* o l'enunciato DECLARE FUNCTION *NomeFunzione(par1,par2,...)* indica che la procedura specificata può essere chiamata dal programma. Il tipi per ciascun parametro viene determinato da un simbolo di dichiarazione. I parametri devono essere dello stesso tipo di quelli definiti nella procedura. Una procedura senza parametri deve apparire in un enunciato DECLARE con una coppia di parentesi vuote. QBasic utilizza DECLARE per verificare che le chiamate al sottoprogramma utilizzino il numero e il tipo appropriato di argomenti. Gli enunciati DECLARE per ciascuna procedura vengono automaticamente inseriti all'inizio del programma quando questo viene salvato. Gli enunciati DECLARE non possono essere inseriti all'interno di una procedura.

DEF FN/END DEF Le funzioni DEF FN definite dall'utente possono essere create in due modi: con una definizione su riga singola nella forma DEF FN *nome(listaParametri)=espressione*, o da un blocco di istruzioni che inizia con l'enunciato DEF FN *nome(listaParametri)*, contiene uno o più comandi che calcolano il valore della funzione e termina con la parola chiave END DEF. L'elenco di variabili specificato in *listaParametri* costituisce l'input della funzione. Se uno degli enunciati nel blocco ha la forma *FNnome=espressione*, la funzione genera un'espressione.

Le funzioni DEF FN devono essere definite prima di essere usate. Ciò significa che devono esser posizionate all'inizio del programma. Le variabili all'interno di un blocco sono globali (vengono cioè condivise dall'intero programma) a meno che non vengano dichiarate come statiche tramite l'enunciato STATIC v*ar1,var2,...,varN*. Le variabili statiche non possono essere usate al di fuori del blocco in cui vengono definite, ma conservano il loro valore anche al termine della funzione. Questo metodo per la creazione di funzioni definite dall'utente è ormai obsoleto, dato che è ora disponibile l'enunciato FUNCTION che supporta la ricorsività, e l'uso degli array e dei record come argomenti.

DEFINT, DEFLNG, DEFSNG, DEFDBL, DEFSTR Si può specificare il tipo di una variabile utilizzando un simbolo di dichiarazione o uno di questi comandi. L'enunciato

DEFINT *lettera* trasforma tutte le variabili che iniziano con la lettera *lettera* per cui non è stato dichiarato un tipo, in variabili intere. Un enunciato nella forma DEFINT *lettera-lettera2* trasforma in variabili intere tutte le variabili che iniziano con una lettera compresa tra *lettera* e *lettera2* per cui non è stato dichiarato un tipo. Gli enunciati DEFLNG, DEFSNG, DEFDBL e DEFSTR operano in modo analogo assegnando, rispettivamente, il tipo intero lungo, a precisione singola, a precisione doppia e stringa.

DEF SEG L'enunciato DEF SEG = *n* specifica che il segmento di memoria corrente consiste delle locazioni di memoria comprese tra 16**n* a 16**n*+65535. Di conseguenza, tutti gli enunciati che accedono direttamente alla memoria, come PEEK, POKE, BLOAD e BSAVE, faranno riferimento alle locazioni di memoria di questo segmento.[9]

DIM L'enunciato DIM *nomeArray*(*m* TO *n*) definisce un array con indice compreso tra *m* e *n* (inclusi), dove *m* e *n* sono numeri interi compresi tra -32768 e 32767. Un enunciato nella forma DIM *nomeArray*(*m* TO *n*,*p* TO *q*) definisce un array bidimensionale (con due indici). Si possono definire in modo analogo degli array con tre indici. Se *m* e *p* sono uguali a zero, gli enunciati DIM sopra riportati possono essere cambiati in DIM *nomeArray*(*n*) e DIM *nomeArray*(*n*;*q*). L'enunciato DIM *nomeVariabile* AS *tipoVariabile*, dove *tipoVariabile* è INTEGER, LONG, SINGLE, DOUBLE, STRING, STRING**n* o un tipo definito dall'utente, specifica il tipo di variabile. La parola chiave SHARED posta alla fine del comando DIM consente di accedere agli array o alle variabili da tutte le procedure.[17]

DO/LOOP Un enunciato nella forma DO, DO WHILE *cond* o DO UNTIL *cond*, viene usato per contrassegnare l'inizio di un blocco di comandi da ripetere. Un enunciato nella forma LOOP, LOOP WHILE *cond* o LOOP UNTIL *cond*, consente di contrassegnare la fine del blocco. Ogni volta che viene incontrato un enunciato contenente la parola chiave WHILE o UNTIL, la veridicità della condizione determina se il programma deve ripetere o meno il blocco di istruzioni. È anche possibile uscire da un ciclo DO tramite l'istruzione EXIT DO.

DRAW L'enunciato grafico DRAW *a$*, dove *a$* è una stringa di direzioni e argomenti, viene usato per tracciare delle figure sullo schermo. La ricchezza e la varietà delle opzioni disponibili per il comando DRAW costituiscono un piccolo linguaggio grafico. L'enunciato DRAW può essere utilizzato per generare delle linee rette che iniziano dall'ultimo punto indirizzato e si estendono in diverse direzioni. Dopo aver tracciato una linea, il punto finale della linea diventa il nuovo ultimo punto indirizzato. Le direzioni possibili sono U (alto), D (basso), L (sinistra), R (destra), E (nordest), F (sudest), G (sudovest) e H (nordovest). Se *Y* è una di queste direzioni e *n* un numero, l'enunciato DRAW "*Yn*" traccia una linea di *n* unità nella direzione *Y*. Se una direzione è preceduta dalla lettera N, l'ultimo punto indirizzato non cambia dopo il disegno della linea. Se una direzione è preceduta da una B, viene tracciata una linea invisibile e l'ultimo punto indirizzato diventa il punto finale di quella linea. Si possono combinare diverse stringhe in un unico enunciato DRAW della forma DRAW "*Yn Zm*...".

Seguono alcuni esempi del comando DRAW:

DRAW "An"	traccia le linee successive ruotate di *n**90 gradi;
DRAW "Cn"	traccia le linee successive usando il colore *n* della palette corrente;
DRAW "M *x,y*"	traccia una linea dall'ultimo punto indirizzato al punto (*x,y*). Se si antepone un segno più o un segno meno a *x* o *y*, vengono usate delle coordinate relative);
DRAW "Pc,B"	riempie la regione contenente l'ultimo punto indirizzato il cui bordo ha colore *b* con il colore *c* della palette corrente;
DRAW "Sn"	imposta la scala a *n*/4 rispetto alla scala originale;
DRAW "TAn"	traccia le linee successive ruotate di *n* gradi.

Gli enunciati DRAW possono usare variabili numeriche per ottenere gli argomenti numerici servendosi della funzione STR$. Ad esempio, l'enunciato DRAW "M 100,25" potrebbe essere scritto come *x*=100:*y*=25:DRAW "M"+STR$(*x*)+","+STR$(*y*).[7, 8, 9]

$DYNAMIC Il metacomando REM $DYNAMIC specifica che, per ogni array dimensionato dopo quel punto, la memoria deve essere riservata dinamicamente. Gli array dinamici possono essere cancellati per liberare dello spazio di memoria o ridimensionati. La memoria per un array viene automaticamente riservata in modo dinamico se l'array è locale in una procedura non statica, se viene dimensionato usando una variabile, o se viene dichiarato in un enunciato COMMON o REDIM.[16, 17]

END L'enunciato END conclude l'esecuzione del programma e chiude tutti i file. Inoltre, gli enunciati END DEF, END FUNCTION, END IF, END SELECT, END SUB e END TYPE vengono usati per indicare, rispettivamente, il termine di una funzione, di blocchi di funzione, di blocchi IF, di blocchi SELECT CASE, di sottoprogrammi, e di dichiarazioni di tipo definite dall'utente.

ENVIRON QBasic dispone di una tabella di ambiente che consiste di equazioni nella forma "*nome-valore*" ereditate dal DOS nel momento in cui viene avviato QBasic. L'enunciato ENVIRON viene usato per modificare questa tabella. Il comando ENVIRON "*nome*=;" rimuove qualsiasi equazione la cui parte sinistra sia uguale a *nome*. Il comando ENVIRON "*nome=valore*" inserisce l'equazione tra virgolette nella tabella.

ENVIRON$ Se *nome* è la parte sinistra di un'equazione nella tabella di ambiente di QBasic, la funzione ENVIRON$("*nome*") restituisce una stringa contenente la parte di destra dell'equazione. Il valore di ENVIRON$(*n*) è la *ennesima* equazione della tabella di ambiente di QBasic.

EOF Si supponga che un file sia stato aperto per l'inserimento con numero di riferimento *n*. La funzione EOF(*n*) restituisce -1 (vero) se è stata raggiunta la fine del file, e 0 in caso contrario.

Nota: La condizione logica NOT EOF(*n*) risulta vera fino a quando non viene raggiunta la fine del file.

Se usata in un file di comunicazione, EOF(*n*) risulta vera se il buffer di comunicazione è vuoto e falsa in caso contrario.

ERASE Nel caso di array statici, ERASE *nomeArray* azzera tutti gli elementi dell'array. Nel caso di array dinamici, invece, ERASE *nomeArray* rimuove l'array dalla memoria.[17, 1]

Nota: Dopo essere stato cancellato, un array dinamico può essere nuovamente dimensionato. Tuttavia, il numero delle dimensioni deve essere uguale a quello precedente.

**ERDEV e ERDEV$** Quando si verifica un errore con una periferica, la funzione ERDEV fornisce alcune informazioni sul tipo di errore e sulla periferica. La funzione ERDEV$ restituisce il nome della periferica. Queste funzioni sono utili nelle routine per la gestione degli errori.[15, 10]

ERR e ERL Quando si verifica un errore durante l'esecuzione di un programma, la funzione ERR restituisce un numero che identifica il tipo di errore, mentre la funzione ERL fornisce il numero della riga di programma in cui si è verificato l'errore. Se l'enunciato che contiene l'errore non ha un numero di riga, viene fornito il numero della riga precedente. Se anche a questa riga non è associato un numero, viene restituito il valore 0. Queste funzioni vengono usate nelle routine di gestione degli errori.[15]

ERROR L'enunciato ERROR *n* simula l'errore identificato dal numero *n*, dove *n* può essere compreso tra 0 e 255. Questa funzione risulta utile come strumento di collaudo.

EXIT Il comando EXIT può essere usato nei cinque formati seguenti: EXIT FOR, EXIT SUB, EXIT FUNCTION, EXIT DEF e EXIT DO. L'enunciato EXIT indica al programma di uscire da un ciclo prematuramente; EXIT FOR esce da un ciclo FOR/NEXT e passa il controllo al'istruzione successiva alla parola chiave NEXT, mentre EXIT SUB consente di uscire da un sottoprogramma e ritornare all'enunciato successivo al comando CALL. Gli altri formati dell'enunciato EXIT operano in modo analogo.

EXP La funzione EXP(*x*) restituisce e^x, dove e (circa 2,71828) è la base del logaritmo naturale.

FIELD Se si utilizza il metodo del buffer per accedere a un file ad accesso casuale, un enunciato nella forma FIELD *#n,w1* AS *strvar1,w2* AS *strvar2,...* partiziona ogni record del file con numero di riferimento *n* nei campi di lunghezza *w&, w2,...* e con nome *strvar1, strvar2...* La somma di *w1+w2+...* generalmente è uguale, e non deve esser superiore, alla lunghezza del record specificata al momento dell'apertura del file. L'enunciato GET assegna i valori direttamente alle variabili stringhe *stravar1, strvar2,....*

FILEATTR Dopo aver aperto un file con numero di riferimento *n*, la funzione FILEATTR (*n*,1) restituisce 1, 2, 4, 8 o 32 a seconda del fatto che il file sia stato aperto, rispettivamente, con la specifica INPUT, OUTPUT, APPEND, RANDOM o BINARY. La funzione FILEATTR (*n*,2) fornisce il puntatore di file del DOS, un numero che identifica in maniera univoca il file e viene usato nella programmazione in linguaggio assembly.

FILES Il comando FILES *percorso* visualizza i nomi dei file contenuti nella directory specificata da *percorso*. Se *percorso* non viene incluso, FILES mostra i file contenuti nella directory corrente dell'unità disco attiva.[3]

FIX La funzione FIX(*x*) restituisce il numero intero ottenuto rimuovendo la parte decimale di *x*.

FOR/NEXT L'enunciato FOR *indice*=*a* TO *b* STEP *s* imposta il valore della variabile *indice* su *a* e ripete i comandi compresi tra FOR e NEXT. Ogni volta che viene raggiunto il comando NEXT, *s* viene aggiunto a *indice*. La procedura continua fino a quando il valore di *indice* supera *b*. Benché i numeri *a*, *b* e *s* possano essere di un qualsiasi tipo, il ciclo viene eseguito più rapidamente al diminuire della precisione. L'enunciato FOR *indice* = *a* TO *b* equivale all'enunciato FOR *indice*=*a* TO *b* STEP 1. L'indice di seguito alla parola chiave NEXT è opzionale.

FRE In qualsiasi momento, la funzione FRE(" ") o FRE(0) restituisce la quantità di memoria disponibile per la memorizzazione di nuovi dati stringa. La funzione FRE(-1) fornisce la memoria disponibile per i nuovi array numerici. Questa funzione è utile per determinare se la memoria disponibile è sufficiente o meno per contenere nuovi array. La funzione FRE(-2) restituisce la quantità più piccola di spazio nello stack verificatasi durante l'esecuzione del programma. Per *n* diverso da -1 o -2, FRE(*n*) fornisce lo stesso valore di FRE(0).

FREEFILE Quando si apre un file, viene sempre assegnato un numero di riferimento. La funzione FREEFILE restituisce il successivo numero di riferimento disponibile.

FUNCTION Una funzione è un blocco composto da più righe che inizia con il comando FUNCTION *NomeFunzione*(*listaParametri*), è seguito da una o più istruzioni che svolgono le operazioni delle funzioni, e termina con l'enunciato END FUNCTION. La lista di parametri, *listaParametri*, è un elenco di variabili attraverso cui vengono passati dei valori alla funzione quando questa viene richiamata. I parametri possono essere di tipo numerico, stringa, record definiti dall'utente o array. Per specificare il tipo dei parametri si possono usare i simboli di dichiarazione, gli enunciati DEFtipo o la parola chiave AS. I nomi degli array che appaiono nella lista dei parametri dovrebbero essere seguiti da una coppia di parentesi vuote. Le funzioni vengono denominate seguendo le stesse convenzioni applicate alle variabili, ad eccezione del fatto che i nomi non possono iniziare con FN. Il valore di un argomento usato in una chiamata di funzione può essere modificato dalla funzione a meno che la variabile non sia racchiusa tra parentesi. Le variabili sono locali a meno che non

vengano dichiarate con l'enunciato STATIC o SHARED. Un enunciato nella forma FUNCTION *NomeFunzione*(*listaParametri*) STATIC indica al programma di considerare tutte le variabili nella funzione come statiche; ciò significa che queste variabili non possono essere usate fuori dalla funzione, ma conservano il loro valore. Le funzioni possono richiamare se stesse (*ricorsività*) o altre procedure. Tuttavia, non si può definire nessuna procedura all'interno di una funzione.

GET (file) I record definiti dall'utente mettono a disposizione un metodo efficiente per la gestione dei file ad accesso casuale. Dopo aver definito un record ed aver dichiarato una variabile dello stesso tipo, ad esempio *recVar*, il file viene aperto con una lunghezza uguale a LEN(*recVar*). Il record numero *r* del file ad accesso casuale viene letto e assegnato alla variabile *recVar* tramite l'enunciato GET *#n,r,recVar*.

L'enunciato GET può essere usato anche per prelevare dei dati da un file binario. Si supponga che *var* sia una variabile contenente un valore composto da *b* byte. (Ad esempio, se *var* è una variabile intera, *b* sarà uguale a 2. Se *var* è una normale variabile stringa, *b* sarà uguale alla lunghezza della stringa ad essa assegnata). L'enunciato GET *#n,p,var* assegna alla variabile *var* i *b* byte consecutivi partendo da byte in posizione *p* nel file con numero di riferimento *n*.

Nota: Le posizioni sono numerate 1, 2, 3, ...

Se *p* viene omesso, la lettura incomincia dall'inizio del file.

Se si utilizza il metodo del buffer per accedere a un file ad accesso casuale, l'enunciato GET *#n,r* carica il record numero *r* dal file con numero di riferimento *n* e assegna i valori del record alle variabili specificate nell'enunciato FIELD. Se *r* viene omesso, viene letto il record successivo all'ultimo elaborato da un comando GET o PUT.[11, 12]

GET (grafici) Un enunciato grafico nella forma GET (*x1,y1*)-(*x2,y2*),*nomeArray* memorizza una descrizione della porzione rettangolare dello schermo i cui angoli opposti sono (*x1,y1*) e (*x2,y2*) nell'array *nomeArray*. La regione rettangolare può essere duplicata in un'altra posizione dello schermo usando un comando PUT. Gli enunciati GET e PUT vengono spesso utilizzati per generare delle animazioni.[8]

GOSUB Il comando GOSUB *etichetta* consente di passare il controllo del programma all'istruzione associata a *etichetta*. Quando viene raggiunto un comando RETURN, il programma ritorna all'istruzione successiva al comando GOSUB.[13, 14]

Nota: Sia l'enunciato GOSUB che l'etichetta a cui fa riferimento devono trovarsi nella stessa parte del programma (nella porzione principale o nella stessa procedura).

GOTO Il comando GOTO *etichetta* passa il controllo all'istruzione che si trova immediatamente dopo l'*etichetta* specificata.[13]

Nota: Sia l'enunciato GOTO che l'etichetta a cui fa riferimento devono trovarsi nella stessa parte del programma.

HEX$ Se *n* è un numero intero compreso tra 0 e 2.147.483.647, la funzione HEX$(*n*) restituisce una stringa composta dalla rappresentazione esadecimale di *n*.

IF (riga singola) Un enunciato nella forma IF *condizione* THEN *comando* indica al programma di eseguire *comando* se *condizione* è vera. In caso contrario, l'esecuzione prosegue con l'istruzione successiva. Un enunciato nella forma IF *condizione* THEN *comando1* ELSE *comando2* indica al programma di eseguire *comando1* se *condizione* è vera e *comando2* se è falsa.

IF (blocco) Un blocco di istruzioni che inizia con l'enunciato IF *condizione* THEN e termina con l'enunciato END IF, indica al programma di eseguire tutti i comandi compresi nel blocco solo se la *condizione* risulta vera. Se il gruppo di comandi è diviso in due parti da un enunciato ELSE, la prima parte viene eseguita se la *condizione* è vera, mentre la seconda solo se la *condizione* è falsa. Si può anche utilizzare la parola chiave ELSEIF per nidificare più enunciati.

**INKEY$** L'enunciato *a$*=INKEY$ assegna alla variabile *a$* i(l) carattere(i) correntemente contenuto(i) nel buffer di tastiera. I numeri, le lettere e i simboli sono identificati da un singolo carattere, mentre tasti come F1, Home e Ins sono identificati da due caratteri, CHR$(0) seguito da CHR$(*n*), dove *n* è lo scan code associato al tasto. Se il buffer non contiene nessun carattere, viene restituita una stringa nulla.

INP La funzione INP(*n*) restituisce il byte letto dalla porta *n*.[10]

INPUT Un enunciato nella forma INPUT *var* indica al computer di visualizzare un punto di domanda e di attendere una risposta da memorizzare nella variabile *var*. Gli enunciati nella forma INPUT "*richiesta*"; *var* inseriscono il messaggio *richiesta* prima del punto di domanda, gli enunciati INPUT "*richiesta*", *var* visualizzano il messaggio senza punto di domanda e gli enunciati INPUT; *var* rimuovono il carattere di ritorno a capo che segue l'inserimento. In ciascuno degli enunciati sopra citati, *var* può essere sostituito da un numero di variabili separate da virgole. Quando l'utente inserisce il numero appropriato di valori separati da virgole e preme Invio, ciascun valore viene assegnato alla variabile corrispondente.

INPUT# L'enunciato INPUT *#n,var* legge l'elemento successivo nel file sequenziale con numero di riferimento n e lo assegna alla variabile *var*. L'enunciato INPUT *#n,var1,var2,...,varN* legge una serie di valori e li assegna alle variabili specificate.

**INPUT$** Un enunciato nella forma *a$*=INPUT$(*n*) indica al programma di fermarsi fino a quando l'utente non ha digitato *n* caratteri. La stringa composta da questi *n* caratteri viene assegnata alla variabile *a$*. L'enunciato *a$*=INPUT$(*n,m*) assegna gli *n* caratteri successivi prelevati dal file con numero di riferimento *m* alla variabile *a$*.

INSTR La funzione INSTR(*a$*, *b$*) restituisce la posizione in cui *b$* è contenuta in *a$*. Il valore di INSTR(*n*, *a$*, *b$*) equivale alla prima posizione in cui è contenuta *b$* partendo dal carattere *n* di *a$*. Se *b$* non è una sottostringa di *a$*, viene fornito il valore 0.

INT La funzione INT(*x*) restituisce il numero intero più grande minore o uguale a *x*.

IOCTL e IOCTL$ Dopo aver attivato una periferica con numero di riferimento *n*, la funzione IOCTL$(*n*) fornisce la stringa di controllo letta dalla periferica, mentre l'enunciato IOCTL #*n*, *a$* invia la stringa *a$* alla periferica.[10]

KEY Il comando KEY *n*, *a$* assegna la stringa *a$* al tasto funzione F*n*. La lunghezza della stringa *a$* non può superare i 15 caratteri, mentre *n* può essere compreso tra 1 e 10 (si possono anche assegnare i numeri 30 e 31 per indicare, rispettivamente, i tasti funzione F11 e F12). Dopo aver impartito il comando KEY *n*, *a$*, la pressione del tasto F*n* riproduce i caratteri contenuti in *a$*. L'enunciato KEY ON può essere usato per visualizzare nell'ultima riga dello schermo di output i primi sei caratteri delle stringhe assegnate. Il comando KEY OFF disattiva questa visualizzazione. L'enunciato KEY LIST visualizza interamente tutte le stringhe assegnate.

KEY(*n*) L'enunciato ON KEY(*n*) GOSUB *etichetta* consente di rilevare la pressione di un tasto funzione. Dopo aver impartito questo comando, ogni volta che si preme il tasto funzione F*n*, il controllo viene passato alla subroutine che inizia con *etichetta*. Questa funzione viene disabilitata con KEY(*n*) OFF e sospesa con KEY(*n*) STOP. La subroutine che inizia nella posizione *etichetta* deve trovarsi all'interno del programma principale.[15, 13, 14]

KILL L'enunciato KILL *specFile* cancella dal disco il file specificato.[4]

LBOUND Per quanto riguarda un array a una dimensione, la funzione LBOUND(*nomeArray*) restituisce l'indice più piccolo che può essere utilizzato. Per qualsiasi altro array, la funzione LBOUND(*nomeArray*, *n*) fornisce il valore più piccolo che può essere usato per l'indice *n* dell'array. Ad esempio, dopo aver impartito il comando DIM esempio(1 TO 31,1 TO 12,1990 TO 1999), la funzione LBOUND(esempio,3) restituisce il valore più piccolo utilizzabile per il terzo indice di esempio(), cioè 1990.

**LCASE$** La funzione LCASE$(*a$*) converte tutti i caratteri contenuti in *a$* in lettere minuscole.

**LEFT$** La funzione LEFT$(*a$*, *n*) restituisce una stringa composta dai primi *n* caratteri di *a$*. Se *n* è maggiore della lunghezza di *a$*, la stringa fornita da LEFT$ sarà uguale ad *a$*.

LEN La funzione LEN(*a$*) restituisce il numero di caratteri presente in *a$*. Se *var* non è una stringa a lunghezza variabile, la funzione LEN(*var*) fornisce il numero di byte

necessari per contenere il valore della variabile in memoria. Ciò significa che LEN(*var*) restituisce 2, 4, 4 o 8, rispettivamente per le variabili intere, intere lunghe, a precisione singola e a precisione doppia. Quando *var* è un tipo di record definito dall'utente, LEN(*var*) fornisce il numero di byte necessari per memorizzare il valore della variabile.

LET L'enunciato LET *var=espressione* assegna il valore dell'*espressione* alla variabile *var*. Se *var* è una variabile di lunghezza fissa *n* e LEN(*espressione*) è maggiore di *n*, vengono assegnati a *var* solo i primi *n* caratteri dell'*espressione*. Se LEN(*espressione*)<*n*, *espressione* viene assegnata a *var* e vengono aggiunti gli spazi necessari per raggiungere la lunghezza specificata. Se *var* è un tipo definito dall'utente, *espressione* deve essere dello stesso tipo. L'enunciato *var=espressione* equivale a LET *var=espressione*.

LINE L'enunciato grafico LINE (*x1,y1*)-(*x2,y2*) traccia una linea che collega i due punti. Se il primo punto viene omesso, la linea si estende all'ultimo punto indirizzato a quello specificato. Se viene impartito il comando LINE (*x1,y1*)-(*x2,y2*),*c*, alla linea viene assegnato il colore *c* della palette corrente. L'enunciato LINE (*x1,y1*)-(*x2,y2*),,*B* traccia un rettangolo che ha, come vertici opposti, i punti specificati. Se *B* viene sostituito da *BF*, viene tracciato un rettangolo pieno. Se *s* è un numero esadecimale compreso tra 0 e &HFFFF, l'enunciato LINE (*x1,y1*)-(*x2,y2*),,,*s* traccia una linea con stile *s* dal punto (*x1,y1*) al punto (*x2,y2*).[7,8,6]

LINE INPUT Gli enunciati LINE INPUT *a$*, LINE INPUT "*richiesta*";*a$* e LINE INPUT "*richiesta*",*a$* sono simili agli enunciati INPUT corrispondenti. Tuttavia, l'utente può rispondere con una qualsiasi stringa che può contenere anche delle virgole, degli spazi in testa e delle virgolette. L'intera stringa viene assegnata alla variabile *a$*.

LINE INPUT# Dopo aver aperto un file sequenziale per la lettura con il numero di riferimento *n*, l'enunciato INPUT LINE #*n*,*a$* assegna alla variabile *a$* tutti i caratteri che si trovano dalla posizione corrente del puntatore nel file alla prima coppia di CR/LF.

LOC Questa funzione restituisce la posizione corrente del puntatore in un file sequenziale, ad accesso casuale e binario. Per quanto riguarda un file sequenziale con numero di riferimento *n*, LOC(*n*) equivale al numero di blocchi di 128 caratteri letti o scritti nel file dalla sua apertura. Nel caso di file ad accesso casuale, LOC(*n*) fornisce il numero del record corrente (l'ultimo record letto o scritto, o quello raggiunto tramite l'enunciato SEEK). Nei file binari, invece, LOC(*n*) corrisponde al numero di byte, partendo dall'inizio del file, che sono stati letti o scritti. Infine, se usato nelle comunicazioni, LOC(*n*) restituisce il numero di byte in attesa nel buffer di comunicazione con numero di riferimento *n*.[12]

LOCATE L'enunciato LOCATE *r,c* posiziona il cursore sulla riga *r* e sulla colonna *c* nello schermo. Il comando LOCATE,,0 disabilita la visualizzazione del cursore, mentre LOCATE,,1 la riattiva. Se *n* e *m* sono numeri interi compresi tra 0 e 31, il comando LOCATE,,,*m,n* cambia la dimensione del cursore.

LOCK Il comando LOCK viene utilizzato nei programmi che operano in rete. Il comando SHARE del DOS abilita la condivisione dei file e dovrebbe essere impartito dal DOS prima di eseguire un qualsiasi comando LOCK. Dopo aver aperto un file con numero di riferimento *n*, l'enunciato LOCK *#n* impedisce l'accesso al file *n* per qualsiasi altra procedura. Nel caso dei file ad accesso casuale, l'enunciato LOCK *#n, r1* TO *r2* impedisce l'accesso ai record compresi tra *r1* e *r2*. Se è stato aperto un file binario, lo stesso enunciato impedisce l'accesso ai byte compresi tra *r1* e *r2*. Quando si lavoro con dei file sequenziali, tutti i formati del comando LOCK generano lo stesso effetto di LOCK *#n*. Il comando UNLOCK viene usato per rimuovere i blocchi dai file. Si dovrebbero rimuovere tutti i blocchi prima di chiudere un file e prima della fine del programma.[12]

LOF Dopo aver aperto un file con il numero di riferimento *n*, la funzione LOF(*n*) riporta il numero di caratteri contenuti nel file (cioè, la lunghezza del file). Per quanto riguarda le comunicazioni, LOF(*n*) restituisce il numero di byte presenti nel buffer di comunicazione con numero di riferimento *n*.

LOG Se *x* è un numero positivo, il valore di LOG(*x*) è il logaritmo naturale (in base e) di *x*.

LPOS Le stampanti dispongono di un buffer che contiene i caratteri che devono essere stampati. La funzione LPOS(1) fornisce la posizione corrente nel buffer per la prima stampante (LPT1), mentre LPOS(2) fornisce la posizione corrente nel buffer per la seconda stampante (LPT2).

LPRINT e LPRINT USING Questi enunciati inviano i dati alla stampante nello stesso modo in cui PRINT e PRINT USING li inviano sullo schermo. Inoltre, LPRINT può essere usato per impostare varie modalità di stampa, come la larghezza dei caratteri e la spaziatura tra le righe.

LSET Se *a$* è una variabile di campo di un file ad accesso casuale, l'enunciato LSET *af$=b$* assegna la stringa *b$*, eventualmente troncata o completata con degli spazi, a *af$*. Se *a$* è una variabile normale, l'enunciato LSET *a$=b$* sostituisce il valore di *a$* con una stringa della stessa lunghezza specificata da *b$*, eventualmente troncata o completata con degli spazi. LSET può anche essere usato per assegnare un record definito dall'utente a un altro record di tipo differente.[11]

**LTRIM$** La funzione LTRIM$(*a$*) restituisce la stringa ottenuta rimuovendo tutti gli spazi in testa dalla stringa *a$*. La stringa *a$* può essere di lunghezza fissa o variabile.

**MID$** La funzione MID$(*a$,m,n*) restituisce la sottostringa di *a$* che inizia al carattere *m* di *a$* e contiene gli *n* caratteri successivi. Se il parametro *n* viene omesso, MID$(*a$,m*) fornisce una stringa che contiene tutti i caratteri di *a$* a partire dal carattere *m*. L'enunciato MID$(*a$,m,n*)=*b$* sostituisce i caratteri in *a$* che iniziano dal carattere *m* con i primi *n* caratteri della stringa *b$*.

MKDIR Il comando MKDIR*percorso\ nomeDir* crea una sottodirectory denominata *nomeDir* nella directory specificata da *percorso*.[3]

MKI$, MKL$, MKS$, MKD$ Queste funzioni convertono, rispettivamente, dei numeri interi, interi lunghi, a precisione singola e a precisione doppia, in stringhe di lunghezza 2, 4, 4 e 8. Questa conversione è necessaria quando si utilizza il metodo del buffer con i file ad accesso casuale.[11]

MKSMBF$, MKDMBF$ Quando si utilizza il metodo del buffer con i file ad accesso casuale, queste funzioni convertono, rispettivamente, i numeri a precisione singola e doppia nelle stringhe di lunghezza 4 e 8, il formato binario della Microsoft. Questa conversione è necessaria prima di inserire dei numeri in file ad accesso casuale che devono essere letti con il BASIC Microsoft, GW-BASIC, BASICA o le versioni precedenti di QuickBASIC.1[1]

NAME L'enunciato NAME *specifiche1* AS *specifiche2* viene usato per cambiare il nome e/o la directory di *specifiche1* nel nome e/o directory specificata da *specifiche2.* Le due specifiche devono riferirsi alla stessa unità disco.[4]

OCT$ Se *n* è un numero intero compreso tra 0 e 2.147.483.647, la funzione OCT$(*n*) restituisce la rappresentazione ottale (in base 8) di *n*.

ON COM(*n*) Se *n* è un numero uguale a 1 o 2, l'enunciato ON COM(*n*) GOSUB *etichetta* consente di rilevare l'attività della porta di comunicazione *n*. Dopo aver eseguito il comando COM(*n*) ON, nel momento in cui arrivano delle informazioni alla porta *n* il controllo del programma passa alla subroutine *etichetta.* Questa subroutine deve trovarsi nella porzione principale del programma, o dopo l'enunciato END.[15, 10, 13, 14]

ON ERROR L'enunciato ON ERROR GOTO *etichetta* consente di impostare una routine per la gestione degli errori. Quando si verifica un errore, il controllo passa alla subroutine *etichetta*. Questa subroutine deve trovarsi nella porzione principale del programma. Si veda RESUME per ulteriori informazioni.[15, 13]

ON...GOSUB e ON...GOTO L'enunciato ON espressione GOSUB *etichetta1, etichetta2,...* indica al programma di passare il controllo alla subroutine *etichetta1, etichetta2...* a seconda del valore di espressione, che deve essere 1, 2, ... L'enunciato ON...GOTO opera in modo analogo. Gli enunciati GOSUB e GOTO e le loro destinazioni devono trovarsi nella stessa parte del programma, sia nella porzione principale che in una procedura.[13]

ON KEY(*n*) L'enunciato ON KEY(*n*) GOSUB *etichetta* consente di rilevare la pressione di un tasto funzione. Dopo aver impartito il comando KEY(*n*) ON, ogni volta che si preme il tasto funzione F*n*, il controllo viene passato alla subroutine che inizia con *etichetta.* Questa subroutine deve trovarsi all'interno del programma principale.[15, 13, 14]

ON PEN L'enunciato ON PEN GOSUB *etichetta* consente di rilevare l'attività di una penna ottica. Dopo aver eseguito il comando PEN ON, nel momento in cui viene utilizzata la penna ottica il controllo del programma passa alla subroutine *etichetta*. Questa subroutine deve trovarsi all'interno del programma principale o dopo l'enunciato END.[15, 13, 14]

ON PLAY(*n*) Il buffer per la musica in background contiene le note, specificate tramite il comando PLAY, che devono essere riprodotte. Se *n* è un numero intero, l'enunciato ON PLAY(*n*) GOSUB *etichetta* consente di rilevare l'attività del buffer per la musica. Dopo aver impartito il comando PLAY ON, non appena il numero di note nel buffer diventa inferiore a *n*, il programma passa il controllo alla subroutine *etichetta*. La capacità di questo buffer è di 32 note, contando le pause eventualmente impostate. La subroutine *etichetta* deve trovarsi all'interno del programma principale o dopo l'enunciato END.[15, 13, 14]

ON STRIG(*n*) Se *n* è 0, 2, 4 o 6, l'enunciato ON STRIG(*n*) GOSUB *etichetta* consente di rilevare lo stato di uno dei pulsanti del joystick. I numeri 0 e 4 sono associati al pulsante inferiore e superiore del primo joystick, mentre i numeri 2 e 6 sono associati al pulsante inferiore e superiore del secondo joystick. Dopo aver impartito il comando STRIG(*n*) ON, il programma passa il controllo alla subroutine *etichetta* ogni volta che viene premuto il pulsante associato al numero *n*. La subroutine *etichetta* deve trovarsi all'interno del programma principale o dopo l'enunciato END.[15, 13, 14]

ON TIMER Se *n* è un intero compreso tra 1 e 86400 (da un secondo a 24 ore), l'enunciato ON TIMER(*n*) GOSUB *etichetta* consente di controllare l'orologio interno del computer. Dopo aver impartito il comando TIMER ON, il programma passa il controllo alla subroutine *etichetta* dopo ogni *n* secondi. La subroutine *etichetta* deve trovarsi all'interno del programma principale o dopo l'enunciato END.[15, 13, 14]

OPEN L'enunciato OPEN *specFile* FOR *modo* AS #*n* consente di accedere al file *specFile* in una delle seguenti modalità: INPUT (lettura da un file sequenziale), OUTPUT (creazione e scrittura di un file sequenziale), APPEND (aggiunta di dati a un file sequenziale) o BINARY (lettura o scrittura di dati in modo arbitrario). L'enunciato OPEN *specFile* FOR RANDOM AS #*n* LEN = *g* consente di gestire il file *specFile* ad accesso casuale in cui ogni record ha lunghezza *g*. Durante l'esecuzione di tutto il programma, si può accedere a un file tramite il numero di riferimento *n* (compreso tra 1 e 255). Alcune varianti del comando OPEN sono: OPEN "SCRN" FOR OUTPUT AS #*n*, OPEN "LPT1" FOR OUTPUT AS #*n* e OPEN "KEYBD" FOR INPUT AS #*n*. Questi enunciati permettono, rispettivamente, di accedere allo schermo, alla stampante e alla tastiera come se si trattasse di file sequenziali.[4, 12]

Il DOS 3.0 e le versioni successive supportano le reti e rendono possibili due nuovi utilizzi del comando OPEN. Il comando SHARE del DOS abilita la condivisione dei file e dovrebbe essere eseguito dal DOS prima di impartire un comando OPEN per la rete. QBasic accede ai file di dati in due modi: lettura o scrittura. Quando diverse procedure possono accedere allo stesso file contemporaneamente, è necessario impostare una

serie di blocchi che impediscano l'accesso a qualsiasi utente ad eccezione di quello che ha aperto il file. L'enunciato OPEN *specFile* FOR *modo* LOCK READ AS #*n*, oppure OPEN *specFile* FOR RANDOM LOCK READ AS #*n* LEN=*g*, apre il file specificato e impedisce a qualsiasi altro utente di leggere quel file fintanto che resta aperto. LOCK WRITE blocca il file solamente in scrittura, mentre LOCK READ WRITE blocca interamente il file. L'opzione LOCK SHARED fornisce un accesso completo al file. Ad eccezione di LOCK SHARED, se un file è stato bloccato per una determinata modalità di accesso, qualsiasi tentativo di azione sul quel file causa la visualizzazione di un messaggio di errore.

OPEN "COM... Se *n* è uguale a 1 o 2, l'enunciato OPEN "COM*n:b,p,d,s,L*" AS #*m* LEN=*g* consente di accedere alla porta seriale *n* usando il numero di riferimento *m* e specificando una dimensione di blocco pari a *g*, una velocità di trasmissione *b*, parità *p*, numero di bit per i dati da usare per trasmettere ciascun carattere *d*, numero di bit di stop *s* e parametri di riga *L*.

OPTION BASE Dopo aver impartito il comando OPTION BASE *m*, dove *m* è uguale a 0 o 1, un enunciato nella forma DIM *nomeArray*(*n*) definisce un array con indice compreso tra *m* e *n*. Grazie a questo comando si possono definire degli array che non utilizzano l'indice numero 0.

OUT L'enunciato OUT *n,m* invia il byte *m* alla porta *n*.[10]

PAINT Se (*x,y*) è un punto non attivo all'interno di una regione dello schermo, il comando PAINT (*x,y*) riempie la regione. In modalità grafica a media risoluzione, se il bordo della regione ha colore *b* della palette corrente, l'enunciato PAINT (*x,y*),*c,b* riempie quella regione con il colore *c* della palette corrente. Se *t$* è una stringa non più lunga di 64 caratteri, l'enunciato PAINT (*x,y*),*t$* riempie la regione con il motivo di riempimento determinato da *t$*.[8, 6]

PALETTE e PALETTE USING Quando un monitor grafico è collegato a una scheda EGA, VGA o MCGA, il comando PALETTE carica i colori nei "barattoli" numerati da 0 a 3, da 0 a 15, da 0 a 63 o da 0 a 255. L'enunciato PALETTE *m,n* assegna il colore *n* al barattolo *m*. Il comando PALETTE USING array(0) specifica che ogni barattolo deve essere riempito con il colore memorizzato nell'elemento corrispondente: *array*(0) barattolo 0, *array*(1) barattolo 1, e così via.[5, 6]

PCOPY A seconda della modalità SCREEN utilizzata, l'adattatore video può disporre di memoria extra che può essere utilizzata per lavorare con diverse schermate, chiamate *pagine* (si veda la discussione sul comando SCREEN per i dettagli). Ad esempio, in modalità testo con 80 caratteri per riga, una scheda CGA dispone di quattro pagine. Il comando PCOPY *m, n* copia il contenuto della pagine *m* nella pagina *n*.

PEEK Ogni locazione di memoria contiene un numero compreso tra 0 e 255. Se *n* è un numero compreso tra 0 e 65535, la funzione PEEK(*n*) restituisce il numero contenuto nella locazione a distanza *n* dall'inizio del segmento corrente.

PEN Gli enunciati PEN ON, PEN OFF e PEN STOP consentono, rispettivamente, di attivare, disattivare e sospendere la lettura dello stato della penna ottica. Per ogni *n* compreso tra 0 e 9, la funzione PEN(*n*) restituisce delle informazioni sullo stato della penna ottica.

PLAY (funzione) Il buffer per la musica in background contiene le note, specificate tramite il comando PLAY, che devono essere riprodotte. La funzione PLAY(0) restituisce il numero di note correntemente contenute nel buffer.

PLAY (comando) L'enunciato PLAY *a$*, dove *a$* è una stringa di note e parametri, riproduce delle note musicali. La ricchezza e la varietà di queste stringhe costituiscono un piccolo linguaggio musicale. Una nota può essere identificata da una lettera compresa tra la A e la G, eventualmente seguita da un segno più o meno per indicare, rispettivamente, un diesis o un bemolle. Il parametro P*n* specifica una pausa di 1/*n* di nota. I parametri O, L, T, MF, MB, ML, MS e MN specificano gli attributi delle note successive e vengono a volte combinati con un numero. Ad esempio, il parametro O *n*, dove *n* è un numero compreso tra 0 e 6, specifica l'ottava per le note successive. Il parametro L *n*, con *n* compreso tra 1 e 64, indica la lunghezza delle note successive in termini di 1/*n* (ad esempio, per *n* uguale a 4 viene suonata una nota di un quarto). Il parametro T *n*, con *n* compreso tra 32 e 255, imposta il tempo per le note successive in termini di *n* quarti di nota per minuto. Le impostazioni di default per i parametri O, L e T sono, rispettivamente, 4, 4 e 120. Il parametro MF indica al programma di riprodurre le note associate al comando PLAY prima di eseguire l'enunciato successivo, mentre la ma modalità MB consente di memorizzare fino a 32 note in un'area di memoria temporanea e di riprodurle mentre l'esecuzione del programma continua. I parametri ML (legato) e MS (staccato) incrementano e decrementano la durata delle note; MN riattiva la modalità standard. È possibile inserire il valore di una variabile numerica in un enunciato PLAY convertendo il valore della variabile in stringa tramite la funzione STR$ ed inserendo la stringa risultante nella posizione appropriata usando l'operatore +. Ad esempio, l'enunciato PLAY "D8" può essere scritto come n=8:PLAY"D"+STR$(n).

Tabella G.1 *La funzione PMAP*

n	c	valore di PMAP(c,n)
0	coordinata x naturale	coordinata x fisica
1	coordinata y naturale	coordinata y fisica
2	coordinata x fisica	coordinata x naturale
0	coordinata y fisica	coordinata y naturale

PMAP La funzione grafica PMAP converte le coordinate naturali di un punto in coordinate fisiche e viceversa, come mostrato nella Tabella G.1.[6]

POINT In modalità grafica, la funzione POINT(*x,y*) restituisce il numero del colore associato al punto con coordinate (*x,y*). Se si dispone di una scheda EGA o VGA, la funzione POINT(*x,y*) fornisce il numero della palette assegnata al punto. I valori delle funzioni POINT(0) e POINT(1) indicano la prima e la seconda coordinata fisica dell'ultimo punto indirizzato, mentre i valori di POINT(2) e POINT(3) costituiscono la prima e la seconda coordinata naturale dell'ultimo punto indirizzato.[7, 5, 8, 6]

POKE Ogni locazione di memoria contiene un numero compreso tra 0 e 255. Se *n* è un numero compreso tra 0 e 65535, l'enunciato POKE *n,m* memorizza il numero *m* nella locazione a distanza *n* dall'inizio del segmento corrente.[9]

POS La funzione POS(0) restituisce il numero di colonna su cui è correntemente posizionato il cursore.

PRESET Si veda PSET.

PRINT Il comando PRINT viene usato per visualizzare i dati sullo schermo. L'enunciato PRINT *espressione* visualizza il valore di *espressione* sullo schermo alla posizione corrente del cursore, e sposta il cursore all'inizio della riga successiva. Ai numeri viene aggiunto uno spazio in coda e ai numeri positivi uno spazio in testa. Se l'enunciato è seguito da un punto e virgola o una virgola, il cursore non si sposta sulla riga successiva dopo la visualizzazione, ma rimane, rispettivamente, nella posizione o nella zona successiva. Nello stesso enunciato PRINT si possono includere diverse espressioni separate da un punto e virgola o da una virgola.

PRINT USING L'enunciato PRINT USING *a$;lista espressioni* visualizza i valori delle espressioni nel formato specificato da *a$*. Questo comando può essere usato per allineare correttamente delle colonne di numeri, per aggiungere delle virgole come separatori delle migliaia e delle cifre decimali, e per includere il segno + o -. I numeri vengono formattati tramite i simboli #, +, $, $$, *, **, ^^^^, virgola (,) e punto (.). Le stringhe vengono formattate dai simboli &, ! e \\. Si faccia riferimento alle tabelle G2 e G3.

Nota: Per utilizzare come testo normale uno dei simboli sopra riportati in una stringa di formato, bisogna anteporre un simbolo di sottolineatura (_).

PRINT# e PRINT# USING Dopo aver aperto un file sequenziale con numero di riferimento *n*, gli enunciati PRINT *#n,espressione* e PRINT# *n*,USING *a$;espressione* inseriscono il valore di espressione nel file nello stesso modo in cui PRINT e PRINT USING lo visualizzano sullo schermo.

PSET e PRESET In modalità grafica, l'enunciato PSET(*x,y*) visualizza il punto con coordinate (*x,y*) usando il colore di primo piano, mentre PRESET(*x,y*) lo visualizza con il colore di sfondo. I comandi PSET(*x,y*),*c* e PRESET(*x,y*),*c* visualizzano il punto (*x,y*) usando il colore *c* della palette corrente.[7, 5, 8, 6]

PUT (file) Se si utilizza il metodo del buffer per accedere a un file ad accesso casuale, dopo averlo aperto con numero di riferimento *n* e aver assegnato i valori appropriati alle variabili, l'enunciato PUT *#n,r* inserisce questi valori nel record numero *r* del file con numero di riferimento *n*. Se *r* viene omesso, viene scritto il record successivo all'ultimo elaborato da un comando GET o PUT.

Si supponga che *recVar* sia un record definito dall'utente e che sia stato aperto un file con l'enunciato nella forma OPEN *specFile* FOR RANDOM AS *#n* LEN = LEN(*recVar*). Il comando PUT *#n,r,recVar* inserisce il valore di *recVar* nel record *r* del file.

L'enunciato PUT viene anche usato per inserire dei dati in un file binario. Si supponga che *var* sia una variabile che contenga un valore composto da *b* byte. (Ad esempio, se *var* è una variabile intera, *b* sarà uguale a 2. Se *var* è una normale variabile stringa, *b* sarà uguale alla lunghezza della stringa ad essa assegnata). L'enunciato PUT *#n,p,var* scrive il contenuto di *var* nelle *b* locazioni consecutive partendo dal byte in posizione *p* nel file con numero di riferimento *n*.

Nota: Le posizioni sono numerate 1, 2, 3, ...

Se *p* viene omesso, la scrittura incomincia dall'inizio del file.[11, 12]

PUT (grafici) Dopo aver memorizzato una regione rettangolare nell'array *nomeArray* tramite il comando GET, l'enunciato PUT (*x,y*), *nomeArray*,PSET inserisce una copia esatta della regione rettangolare sullo schermo, usando come angolo superiore di sinistra il punto (*x,y*). Seguono le possibili alternative all'opzione PSET insieme a una breve descrizione.[7, 8]

Un punto nell'immagine risultante diventa bianco nei casi seguenti:

XOR	se il punto già esistente o quello dell'immagine da trasferire è bianco, ma non entrambi.
AND	se in quella posizione esiste già un punto bianco e se anche il punto corrispondente dell'immagine in fase di trasferimento è bianco.
OR	se il punto già esistente e/o quello dell'immagine in corso di trasferimento è bianco.
PRESET	se il punto corrispondente nell'immagine da trasferire è nero.

RANDOMIZE L'enunciato RANDOMIZE TIMER utilizza automaticamente l'orologio interno del computer per cambiare la sequenza dei numeri generati da RND. RANDOMIZE n cambia la sequenza dei numeri casuali sulla base del valore *n*. Se non viene utilizzato il comando RANDOMIZE, la funzione RND genera sempre la stessa sequenza di numeri ogni volta che si esegue il programma.

READ L'enunciato READ *var1,var2,...* assegna a *var1* il primo dato memorizzato in un enunciato DATA, a *var2* il secondo dato, e così via.

Tabella G.2 *Risultati ottenuti eseguendo PRINT USING a$;n*

Simbolo	Descrizione	n	A$	Risultato
#	Indica una cifra in un campo numerico	1234.6 123 123.4 12345	"########" "########" "########" "####"	1235 123 123 %12345
.	Indica il punto decimale	123.4	"######.#"	123.4
,	Indica di visualizzare la virgola come separatore delle migliaia. La virgola viene aggiunta dopo ogni gruppo di tre cifre partendo dal punto decimale	12345	"#######,"	12,345
$	Visualizza il simbolo $ come primo carattere del campo	23.45	"$####.##"	23.45
$$	Visualizza il simbolo $ davanti alla prima cifra	23.45	"$$###.##"	23.45
**	Sostituisce gli spazi in testa con degli asterischi	23.45	"**######"	******23
*	Visualizza un asterisco come primo carattere del campo	23.45	"*#######"	*
^^^^	Visualizza il numero in notazione esponenziale	-12 12345	"##.#^^^^" "#.##^^^^"	1.2E+04 -.12E+01
^^^^^	Visualizza il numero in notazione esponenziale espansa	12345	"#.#^^^^^"	1.2E+004
+	Riserva uno spazio per il segno della variabile	12 -12	"+#######" "######+"	+12 12-

Tabella G.3 *Risultati ottenuti eseguendo PRINT USING a$;x$*

a$	Descrizione	x$	Risultato
&	Visualizza l'intera stringa	"Nebraska"	Nebraska
!	Visualizza la prima lettera della stringa	"Nebraska"	N
\\	Visualizza le prime *n* lettere della stringa (dove ci sono *n*-2 spazi tra le barre rovesciate). In questo caso, *n*=4	"Nebraska"	Nebr

REDIM L'enunciato REDIM *nomeArray*(...) rimuove l'array specificato dalla memoria e lo ricrea. Le informazioni all'interno delle parentesi hanno la stessa forma e generano lo stesso risultato di quelle usate nel comando DIM. Dopo aver ridimensionato un array, tutti gli elementi vengono riportati al loro valore di default. Benché gli intervalli degli indici possano cambiare, il numero di dimensioni deve rimanere lo stesso dell'array originale. Se si inserisce la clausola SHARED in un enunciato REDIM nella porzione principale del programma, l'array può essere condiviso da tutte le procedure. Si possono ridimensionare solo degli array dinamici.[17, 1]

REM Il comando REM consente di inserire dei commenti in un programma. Una riga nella forma REM *commento* viene ignorata durante l'esecuzione. Un enunciato REM può essere usato anche per inserire un metacomando nel programma e può essere abbreviato con un apostrofo.

RESET Il comando RESET chiude tutti i file aperti. L'uso di RESET equivale al comando CLOSE senza la specifica di un numero di riferimento.

RESTORE Il comando RESTORE *etichetta* indica al programma che alla successiva esecuzione del comando READ deve essere prelevato il primo elemento che si trova nell'enunciato DATA che segue l'*etichetta*. Se il parametro *etichetta* viene omesso, READ accede al primo enunciato DATA presente nel programma. Gli enunciati READ successivi continueranno da quel punto.[13]

RESUME Quando viene incontrato il comando RESUME alla fine di una routine per la gestione dell'errore, il programma riprende l'esecuzione dal punto in cui si è verificato l'errore. Le varianti RESUME *etichetta* e RESUME NEXT indicano al programma di passare, rispettivamente, all'enunciato identificato da *etichetta* o all'enunciato seguente a quello in cui si è verificato l'errore. La combinazione ON ERROR e RESUME NEXT è simile alla combinazione GOSUB e RETURN.[13]

RETURN Quando viene incontrato un comando RETURN alla fine di una subroutine, il programma ritorna all'istruzione seguente a quella che ha originato la chiamata (GOSUB). La variante RETURN *etichetta* indica al programma di passare il controllo all'enunciato identificato da *etichetta*.[13, 14]

**RIGHT$** La funzione RIGHT$(*a$*, *n*) restituisce una stringa composta dagli *n* caratteri più a destra della stringa *a$*. Se *n* è maggiore della lunghezza di *a$*, la stringa fornita da RIGHT$ è uguale ad *a$*.

RMDIR Se *percorso* specifica una directory che non contiene file o altre sottodirectory, il comando RMDIR *percorso* cancella la directory.[3]

RND La funzione RND genera casualmente un numero compreso tra 0 e 1 (1 escluso). Il valore di INT(*n**RND)+1 è un numero intero casuale compreso tra 1 e *n*.

RSET Se *a$* è una variabile di campo di un file ad accesso casuale, il comando RSET *af$=b$* assegna la stringa *b$* a *af$*, eventualmente troncata o completata con degli spazi. Se *a$* è una variabile normale, l'enunciato RSET *a$=b$* sostituisce il valore di *a$* con una stringa della stessa lunghezza specificata da *b$*, eventualmente troncata o completata con degli spazi.[11]

**RTRIM$** La funzione RTRIM$(*a$*) restituisce la stringa ottenuta rimuovendo tutti gli spazi in coda dalla stringa *a$*. La stringa *a$* può essere di lunghezza fissa o variabile.

RUN L'enunciato RUN riavvia il programma corrente. Tutti i valori precedentemente assegnati alle variabili vengono cancellati. La variante RUN *specFile* carica il programma specificato dal disco e lo esegue. Il programma specificato deve essere un programma di QBasic. Il comando RUN *etichetta* riavvia il programma corrente partendo dal punto specificato.[4, 13]

SADD La funzione SADD(*a$*) restituisce la distanza della stringa a lunghezza variabile *a$* in DGROUP, il segmento dati di default.[9]

SCREEN (funzione) La funzione SCREEN(*r,c*) restituisce il codice ASCII del carattere che si trova nella riga *r* e nella colonna *c* dello schermo. Il valore di SCREEN(*r,c*,1) corrisponde al numero del barattolo usato per colorare il carattere.[6]

SCREEN (comando) È possibile attivare una determinata modalità video tramite i comandi riportati nella Tabella G.4.

Tabella G.4 *Enunciati SCREEN*

Enunciato	Modo
SCREEN 0	modalità testo
SCREEN 1	modalità grafica a media risoluzione
SCREEN 2	modalità grafica ad alta risoluzione
SCREEN 3	modalità grafica Hercules due colori, 720x348
SCREEN 7	adattatori EGA, VGA, 16 colori, 320x200
SCREEN 8	adattatori EGA, VGA, 16 colori, 640x200
SCREEN 9	adattatori EGA, VGA, da 4 a 16 colori, 640x350
SCREEN 10	adattatore EGA monocromatico, 640x350
SCREEN 11	adattatori MCGA, VGA, 2 colori, 640x480
SCREEN 12	solo VGA, 16 colori, 640x480
SCREEN 13	adattatori MCGA, VGA, 256 colori, 320x200

Quando si usa un adattatore grafico in modalità testo, il computer può memorizzare il contenuto di alcune schermate differenti, chiamate *pagine*. Il numero di pagine utilizzabili, *n*, dipende dal tipo di adattatore e dalla modalità selezionata. In qualsiasi momento, la pagina correntemente visualizzata è chiamata *visibile*, mentre quella in

fase di scrittura è chiamata *attiva*. Se *a* e *v* sono numeri compresi tra 0 e *n*-1, l'enunciato SCREEN ,,*a,v* definisce la pagina *a* come pagina attiva e la pagina *v* come pagina visibile.[5]

SEEK L'enunciato SEEK *#n,p* imposta la posizione corrente in un file binario o ad accesso casuale con numero di riferimento *n* sul byte o record numero *p*. Dopo aver eseguito questo comando, l'enunciato GET o PUT successivo elabora il byte o il record *p* del file. Il valore della funzione SEEK(*n*) corrisponde alla posizione corrente nel file in termini di byte o di numero di record. Dopo l'esecuzione di un comando GET o PUT, SEEK(*n*) restituisce il numero del byte o del record successivo.[11, 12]

SELECT CASE L'enunciato SELECT CASE mette a disposizione un metodo molto efficace per selezionare il blocco di comandi da eseguire a seconda del valore di un'espressione. Il blocco SELECT CASE inizia con il comando SELECT CASE *espressione* e termina con l'istruzione END SELECT. Tra questi due enunciati ci sono dei comandi nella forma CASE l*istaValori* ed eventualmente ELSE CASE. Gli elementi in *listaValori* possono essere valori singoli, o intervalli di valori nella forma "*a*TO *b*" o "IS<*a*". Ognuno di questi comandi CASE è seguito da un blocco di uno o più enunciati. Il blocco di comandi che segue il primo enunciato CASE *listaValori* in cui il valore di *espressione* viene soddisfatto, è l'unico blocco ad essere eseguito. Se nessuno dei valori in *listaValori* include il risultato di *espressione* ed è presente il comando ELSE CASE, viene eseguito il blocco di enunciati che segue ELSE CASE.

SETMEM L'area di memoria in cui vengono inserite le variabili che non si trovano nel segmento dati di default è denominata heap. La funzione SETMEM consente di modificare e riportare all'impostazione di default la dimensione dell'heap. L'espressione numerica *n* specifica la nuova dimensione (in numero di byte) dell'heap; la dimensione viene incrementata se *n* è un valore positivo, e decrementata in caso contrario. La funzione SETMEM(*n*) restituisce la quantità di memoria nell'heap dopo la modifica.

SGN La funzione SGN(*x*) restituisce 1, 0 o -1 a seconda che *x* sia positivo, uguale a zero, o negativo.

SHARED Un enunciato nella forma SHARED *var1,var2,...* può essere usato all'inizio di una procedura per specificare le variabili che devono essere condivise dall'intero programma. Il tipo di ciascuna variabile viene determinato tramite un simbolo di dichiarazione, un enunciato DEFtipo o una clausola AS. Se viene utilizzato AS in un comando SHARED, è necessario utilizzare un'altra clausola AS per dichiarare il tipo della variabile all'interno del programma principale. Qualsiasi modifica apportata a una variabile condivisa da una procedura cambia il valore della variabile con lo stesso nome nel programma, e viceversa. Quando si dichiara una variabile condivisa in una procedura, è possibile utilizzare la medesima variabile sia nel programma principale che nella procedura senza passarla come argomento. Gli array dimensionati nella porzione principale del programma possono essere condivisi con le procedure, se vengono inclusi in un enunciato SHARED e seguiti da una coppia di parentesi vuote.

SHELL Se *c$* è un comando DOS, l'enunciato SHELL *c$* sospende l'esecuzione del programma in corso, impartisce il comando DOS specificato, e ripassa il controllo al programma. Il comando SHELL usato senza parametri sospende l'esecuzione del programma in corso ed accede al DOS. Per ritornare a QBasic è necessario digitare il comando EXIT dal prompt del DOS.

SIN Per un qualsiasi numero *x*, la funzione trigonometrica SIN(*x*) restituisce il seno dell'angolo di *x* radianti.[2]

SOUND L'enunciato SOUND *f,d* riproduce un suono con una frequenza di *f* hz per la durata di *d**0,055 secondi. Il valore di *f* deve essere almeno di 37.

Nota: Le note generate da un pianoforte hanno una frequenza compresa tra 55 e 8372 hz.

SPACE$ Se *n* è un intero compreso tra 0 e 32767, la funzione SPACE$(*n*) fornisce una stringa composta da *n* spazi.

SPC La funzione SPC viene usata negli enunciati PRINT, LPRINT e PRINT# per generare degli spazi. Ad esempio, il comando PRINT *a$*;SPC(*n*);*b$* inserisce *n* spazi tra le due stringhe.

SQR Per qualsiasi numero non negativo *x*, la funzione SQR(*x*) restituisce la radice quadrata di *x*.

STATIC Un enunciato nella forma STATIC *var1,var2,...* può essere usato all'inizio di una procedura per specificare le variabili che devono essere locali e statiche nella procedura stessa. La memoria per le variabili statiche viene riservata permanentemente e il valore delle variabili rimane inalterato tra una chiamata e l'altra. Il tipo di ciascuna variabile viene determinato tramite un simbolo di dichiarazione, un enunciato DEFtipo o una clausola AS. Le variabili statiche non interferiscono in nessun modo con le variabili dello stesso nome al di fuori della procedura in cui sono state definite. Si possono definire degli array statici includendo, in un enunciato STATIC, i nomi seguiti da una coppia di parentesi vuote.

$STATIC Il metacomando REM $STATIC indica a QBasic di riservare una quantità di memoria statica, o permanente, per tutti gli array specificati negli enunciati DIM successivi. Il metacomando $STATIC inverte l'effetto di $DYNAMIC. Un array viene automaticamente definito come array statico quando viene dimensionato usando delle costanti come indici, quando appare in un enunciato DIM non preceduto da nessun comando di dichiarazione, e quando è dimensionato in una procedura statica.[16, 17]

STICK Per *n*=0 o 1, la funzione STICK(*n*) restituisce, rispettivamente, la coordinata *x* o *y* della leva del primo joystick. Per *n*=2 o 3 vengono fornite le medesime informazioni per il secondo joystick.

STOP Il comando STOP sospende l'esecuzione di un programma. L'esecuzione può essere ripresa dal comando successivo a STOP tramite la pressione del tasto F5.

**STR$** La funzione STR$ converte un numero in stringa. Il valore della funzione STR$(*n*) è la stringa composta dal numero *n* nel formato normalmente visualizzato dal comando PRINT.

STRIG Gli enunciati STRIG ON e STRIG OFF abilitano e disabilitano la lettura dello stato dei pulsanti del joystick. Per ogni *n* compreso tra 0 e 7, la funzione STRIG(*n*) restituisce delle informazioni sullo stato del pulsante del joystick.

STRIG(*n*) L'enunciato ON STRIG(*n*) GOSUB *etichetta* consente di rilevare lo stato di uno dei pulsanti del joystick. I numeri 0 e 4 sono associati al pulsante inferiore e superiore del primo joystick, mentre i numeri 2 e 6 sono associati al pulsante inferiore e superiore del secondo joystick. Dopo aver impartito il comando STRIG(*n*) ON, il programma passa il controllo alla subroutine *etichetta* ogni volta che viene premuto il pulsante associato al numero *n*.[15, 13, 14]

STRING$ Se *n* è un numero intero compreso tra 0 e 32767, la funzione STRING$(*n,a$*) restituisce la stringa composta dal primo carattere di *a$* ripetuto *n* volte. Se *m* è un numero intero compreso tra 0 e 255, STRING$(*n,m*) fornisce la stringa composta dal carattere con codice ASCII *m* ripetuto *n* volte.

SUB/END SUB Un sottoprogramma è un blocco di enunciati che inizia con l'istruzione SUB *NomeSottoprogramma*(*listaParametri*), è seguito da una serie di comandi che svolgono l'opererazione del sottoprogramma, e termina con l'enunciato END SUB. La lista di parametri *listaParametri* contiene le variabili tramite cui vengono passati i valori al sottoprogramma quando questo viene chiamato (si veda il comando CALL). I parametri possono essere numerici, stringhe a lunghezza variabile e array.

SWAP Se *var1* e *var2* sono due variabile dello stesso tipo, il comando SWAP *var1,var2* scambia i valori delle due variabili.

SYSTEM Il comando SYSTEM termina l'esecuzione di un programma, chiude tutti i file e ripassa il controllo a QBasic.

TAB La funzione TAB(*n*) viene usata con i comandi PRINT, LPRINT e PRINT# per spostare il cursore nella posizione *n*. Se *n* è minore della posizione corrente del cursore, il cursore viene spostato nella posizione *n* della riga successiva.

TAN Per qualsiasi numero *x* (ad eccezione di *x*=pi/2, -pi/2, 3*pi/2, -3*pi/2 e così via), la funzione trigonometrica TAN(*x*) restituisce la tangente dell'angolo di *x* radianti.[2]

**TIME$** La funzione TIME$ restituisce l'ora corrente espressa come stringa nella forma oo:mm:ss. Se *t$* è una stringa di questo tipo, il comando TIME$=*t$* imposta l'orologio interno del computer sull'ora corrispondente.

TIMER La funzione TIMER restituisce il numero di secondi passati da mezzanotte rispetto all'ora corrente.

TRON e TROFF Questi enunciati vengono utilizzati per collaudare i programmi. Il comando TRON causa un rallentamento dell'esecuzione del programma ed evidenzia ogni istruzione mano a mano che viene impartita. Il comando TROFF disabilita questa modalità.

TYPE/END TYPE Un blocco di enunciati che inizia con il comando TYPE *nomeTipo* e termina con il comando END TYPE crea un record definito dall'utente. Ogni enunciato all'interno di questo blocco ha la forma *elt* AS *tipo*, dove *elt* è una variabile senza simbolo di dichiarazione e *tipo* può essere INTEGER, LONG, SINGLE, DOUBLE, STRING*n (cioè a lunghezza fissa) o un altro tipo definito dall'utente. Dopo un enunciato nella forma DIM *var* AS *nomeTipo*, l'elemento corrispondente all'enunciato *elt* AS *tipo* viene considerato come *var.elt.* Gli enunciati TYPE non possono apparire all'interno delle procedure.

UBOUND Per quanto riguarda un array a una dimensione, la funzione UBOUND(*nomeArray*) restituisce l'indice più grande che può essere utilizzato. Per qualsiasi altro array, la funzione UBOUND(*nomeArray,n*) fornisce il valore più grande che può essere usato per l'indice *n* dell'array. Ad esempio, dopo aver impartito il comando DIM esempio(1 TO 31,1 TO 12,1990 TO 1999), la funzione UBOUND(esempio,3) restituisce il valore più grande utilizzabile per il terzo indice di esempio(), cioè 1999.

**UCASE$** La funzione UCASE$(*a$*) converte tutti i caratteri contenuti in *a$* in lettere maiuscole.

UNLOCK Il comando UNLOCK viene utilizzato nei programmi che operano in rete. Il comando SHARE del DOS abilita la condivisione dei file e dovrebbe essere impartito dal DOS prima di eseguire un qualsiasi comando LOCK o UNLOCK. Dopo aver usato un comando LOCK per bloccare l'accesso a un determinato file, si dovrebbe usa un comando UNLOCK corrispondente per ripristinare l'accesso. Su supponga di aver aperto un file con numero di riferimento *n*. I blocchi stabiliti tramite gli enunciati LOCK *#n, r1* e LOCK *#n, r1* TO *r2* possono essere rimossi, rispettivamente dagli enunciati UNLOCK *#n, r1* e UNLOCK *#n, r1* TO *r2*. Ci deve essere un'esatta corrispondenza tra gli enunciati LOCK e UNLOCK in un programma; ciò significa che ogni coppia di enunciati LOCK e UNLOCK deve far riferimento allo stesso intervallo di numeri di record o byte.

VAL La funzione VAL viene usata per convertire delle stringhe in numeri. Se i primi caratteri della stringa *a$* sono dei numeri, la funzione VAL(*a$*) restituisce i numeri rappresentati da questi caratteri. Per qualsiasi numero *n*, VAL(STR$(*n*)) è uguale a *n*.

VARPTR e VARSEG Le funzioni VARSEG(*var*) e VARPTR(*var*) restituiscono, rispettivamente, il segmento di memoria e la distanza dall'inizio del segmento in cui

si trova il valore di *var* (se è una stringa numerica o a lunghezza fissa) o il descrittore di *var* (se si tratta di una stringa a lunghezza variabile o di un array).[9]

**VARPTR$** La funzione VARPTR$(*var*) restituisce una stringa di cinque caratteri il cui primo carattere identifica il tipo della variabile e gli altri quattro ne specificano la locazione di memoria. Questa funzione può essere usata in combinazione con DRAW e PLAY.

VIEW L'enunciato grafico VIEW definisce una porzione rettangolare dello schermo come *viewport grafica,* in modo che le figure tracciate successivamente vengano inserite in questa porzione. Ci sono tre varianti del comando VIEW.

In modalità grafica a media risoluzione, la coppia di enunciati WINDOW SCREEN (0,0)-(319,199) e VIEW(*x1,y1*)-(*x2,y2*),*c,b* definisce una viewport il cui angolo superiore di sinistra ha coordinate fisiche (*x1,y1*) e l'angolo inferiore di destra coordinate fisiche (*x2,y2*). Il rettangolo avrà colore di sfondo *c* e un bordo di colore *b*, dove *b* e *c* sono due colori della palette corrente. Gli enunciati grafici successivi utilizzeranno il nuovo sistema di coordinate della viewport come se si trattasse dell'intero schermo. Nelle altre modalità grafiche, i numeri 319 e 199 dovrebbero essere sostituiti dalle coordinate fisiche *x* e *y* del punto nell'angolo in basso a destra dello schermo.

Se non è attivo nessun enunciato WINDOW, il comando VIEW(*x1,y1*)-(*x2,y2*),*c,b* definisce una viewport nella stessa posizione e con gli stessi colori sopra menzionati. Tuttavia, la scala per le figure successive non viene modificata e la parte in eccesso del grafico viene troncata.

Se non è attivo nessun enunciato WINDOW, VIEW SCREEN (*x1,y1*)-(*x2,y2*), *c, b* imposta una viewport nella stessa posizione e con gli stessi colori di quella esaminata all'inizio di questa discussione. Tuttavia, invece di modificare la scala delle figure successive al fine di inserirle interamente nella viewport, viene visualizzata solo la porzione che cade nel rettangolo definito dalla viewport e le parti in eccesso vengono tagliate.[5, 8, 6]

VIEW PRINT Normalmente, lo schermo contiene 25 righe di testo numerate da 1 a 25. Tuttavia, solo le righe da 1 a 24 scorrono. Queste righe sono chiamate viewport di testo. L'enunciato VIEW PRINT *rigaA* TO *rigaB* indica che la viewport di testo consiste delle righe comprese tra *rigaA* e *rigaB*. Dopo l'esecuzione di questo comando, tutto il testo visualizzato dagli enunciati PRINT appare nella viewport e scorrono solamente le righe specificate. Il comando LOCATE è valido solo se il numero di riga specificato è compreso nella viewport di testo corrente, e il comando CLS agisce solamente sulla viewport. Il testo situato nella parte restante dello schermo resta immobile. Il comando VIEW PRINT utilizzato senza parametri include l'intero schermo nella viewport e ha lo stesso effetto di VIEW PRINT 1 TO *h*, dove *h* costituisce il numero di righe di testo dello schermo.

WAIT Se *p* è il numero di una porta, *q* il valore di un byte da ricevere nella porta *p*, e *n* e *m* degli interi compresi tra 0 e 255, l'enunciato WAIT *p,n,m* sospende l'esecuzione del programma finché la condizione ((*q* XOR *m*) AND *n*)<>0 è vera per il byte con valore *q* ricevuto nella porta *p*.[10]

WHILE/WEND Un ciclo WHILE...WEND è costituito da un blocco di enunciati che inizia con il comando WHILE *condizione* e termina con l'istruzione WEND. Dopo aver eseguito un comando WHILE, il programma esegue ripetutamente l'intera sequenza di enunciati contenuti nel blocco finché la condizione resta vera.

WIDTH Quando viene usato con un monitor non monocromatico, il comando WIDTH 40 imposta la visualizzazione del testo su 40 caratteri per riga (la prima zona PRINT contiene 14 posizione e la seconda 26). Il formato standard di 80 caratteri per riga viene ripristinato tramite il comando WIDTH 80 (le prime quattro zone PRINT consistono di 14 posizioni, mentre la quinta di 24 posizioni). In modalità grafica, il comando WIDTH non ha nessun effetto.

Gli adattatori video EGA, VGA e MCGA sono in grado di visualizzare 25, 30, 43, 50 o 60 righe di testo a seconda del tipo di adattatore, del tipo di monitor e della modalità SCREEN. Se *t* è un numero valido per l'adattatore video utilizzato, il comando WIDTH *t* imposta il numero di righe di testo su *t*.

Se *s* è un intero minore di 255, il comando WIDTH "LPT1",*s* forza il comando LPRINT a non stampare più di *s* caratteri per ciascuna riga di testo. QBasic invia alla stampante una coppia di CR/LF dopo ogni gruppo di *s* caratteri, anche se LPRINT non ha terminato la stampa della riga corrente. L'enunciato WIDTH "LPT1",255 specifica una larghezza infinita; ciò significa che la coppia CR/LF viene inviata alla stampante solo quando viene esplicitamente richiesto da LPRINT. Lo stesso effetto può essere ottenuto con il comando WIDTH LPRINT *s*.

WINDOW L'enunciato WINDOW (*x1,y1*)-(*x2,y2*) imposta un sistema di coordinate standard per lo schermo. In questo sistema, le coordinate x si estendono verso destra tra *x1* e *x2* e le coordinate y si estendono verso l'alto tra *y1* e *y2*. I comandi grafici impartiti successivamente a questo comando, tracciano le figure in accordo con questo nuovo sistema di coordinate. Se l'enunciato WINDOW viene sostituito da WINDOW SCREEN, viene impostato un sistema di coordinate non standard. In questo sistema, le coordinate x si estendono verso destra tra *x1* e *x2* e le coordinate y si estendono verso il basso tra *y1* e *y2*.

WRITE Il comando WRITE *esp1,esp2,...* visualizza i valori delle espressioni, uno dopo l'altro, sullo schermo. Le stringhe appaiono racchiuse tra virgolette e i numeri non hanno spazi in testa e in coda. Le virgole vengono visualizzate e non indicano un salto alla zona successiva. Dopo la visualizzazione di tutti i valori, il cursore si sposta all'inizio della riga successiva.

WRITE# Dopo aver aperto un file sequenziale per la scrittura con numero di riferimento *n*, l'enunciato WRITE *#n,esp1,esp2,...* memorizza i valori delle espressioni, uno dopo l'altro, nel file. Le stringhe appaiono racchiuse tra virgolette, i numeri non hanno spazi in testa e in coda, le virgole vengono registrate normalmente, e i caratteri di ritorno a capo e avanzamento riga vengono aggiunti dopo i dati.

NOTE A CONTORNO

1. Valori di default: prima che il programma assegni un valore a una variabile numerica, a una stringa a lunghezza variabile o a una stringa a lunghezza fissa *n*, a queste variabili viene assegnato, rispettivamente, il valore 0, la stringa nulla "" e la stringa di caratteri composta da *n* CHR$(0).
2. Misura in radianti: il sistema di misurazione in radianti misura gli angoli in termini di distanza intorno alla circonferenza del cerchio di raggio 1. Se il vertice di un angolo compreso tra 0 e 360 gradi viene inserito nel centro del cerchio, la lunghezza dell'arco del cerchio contenuto tra i due lati che formano l'angolo è la misura in radianti dell'angolo. Un angolo di *g* gradi ha una misura in radianti uguale a (pi/180)**g*.
3. Directory: si pensi a un disco come a un grosso schedario che contiene delle cartelle, ciascuna delle quali può contenere altre cartelle. Ogni cartella ha un nome unico e viene identificata da un *percorso*: una stringa composta da una lettera dell'unità disco, il segno di due punti (:), una barra rovesciata e il nome della cartella da identificare (eventualmente seguito da altri nomi di sottocartelle separati da barre rovesciate). Ad esempio, il percorso "C:\DAVIDE\GIOCHI" identifica la cartella GIOCHI contenuta nella cartella DAVIDE che è a sua volta contenuta nella cartella principale nell'unità disco C.
   Ogni cartella è chiamata *directory* e la cartella principale *directory radice*. Quando si apre una cartella, le sottocartelle eventualmente presenti vengono chiamate *sottodirectory* (o subdirectory). A questo punto, si pensi a un file come a un foglio di carta inserito in una cartella. Quindi, ogni directory contiene dei file e delle sottodirectory.
   In qualsiasi momento, c'è sempre una (e una sola) *directory corrente*. Inizialmente, la directory corrente è la directory radice. La directory corrente può essere cambiate dal DOS tramite il comando CD o da QBasic con il comando CHDIR. I comandi del DOS o di QBasic che consentono di accedere ai file, come DIR e FILES, agiscono sui file contenuti nella directory corrente a meno che non venga specificato diversamente.
   L'*unità disco di default* è quella indicata dal prompt del DOS. Se in un percorso non viene specificata l'unità disco, viene utilizzata quella di default.
4. Specfile: le specifiche di un file consistono di una stringa composta dalla lettera dell'unità disco, il segno di due punti e il nome del file. Se vengono utilizzate delle directory, il nome del file viene preceduto dal percorso appropriato.

5. Colori per i monitor CGA: con i monitor standard collegati a una scheda grafica CGA sono disponibili 16 colori differenti, identificati dai numeri compresi tra 0 e 15.

0	Nero	4	Rosso	8	Grigio	12	Rosso chiaro
1	Blu	5	Magenta	9	Blu chiaro	13	Magenta chiaro
2	Verde	6	Marrone	10	Verde chiaro	14	Giallo
3	Ciano	7	Bianco	11	Ciano chiaro	15	Bianco intenso

In modalità testo, che viene attivata dal comando SCREEN 0, tutti questi colori sono disponibili per il colore di primo piano, e i primi otto per quello di sfondo. In modalità grafica a media risoluzione, richiamata tramite l'enunciato SCREEN 1, sono disponibili due palette da quattro colori ciascuna.

Palette 0:	0. Colore di sfondo	1. Verde	2. Rosso	3. Marrone
Palette 1:	0. Colore di sfondo	1. Ciano	2. Magenta	3. Bianco

In modalità ad alta risoluzione, richiamata dall'enunciato SCREEN 2, sono disponibili solo due colori: nero (0) e bianco (1). Si veda il Capitolo 8 per ulteriori informazioni sui colori disponibili.

6. Palette: si può pensare a una palette come a una serie di barattoli di vernice. Il numero dei barattoli dipende dalla modalità SCREEN attiva. Benché i barattoli contengano dei colori di default specifici, gli adattatori EGA, MCGA e VGA consentono di cambiare questi colori tramite il comando PALETTE. Enunciati nella forma PSET(*x,y*),*c* e CIRCLE(*x,y*),*c* utilizzano il colore del barattolo *c*. Se non viene impartito un comando COLOR, l'enunciato PRINT visualizza i caratteri utilizzando il colore nel barattolo 0 per lo sfondo e il colore nel barattolo cui è associato il numero più alto per il primo piano.

7. Ultimo punto indirizzato: dopo aver eseguito un qualunque comando grafico, esiste sempre un punto sullo schermo conosciuto come *ultimo punto indirizzato*. Questo è l'ultimo punto a cui un enunciato grafico ha fatto riferimento. Quando si esegue per la prima volta un programma, o si impartisce un comando SCREEN o CLS, l'ultimo punto indirizzato diventa il centro dello schermo. Ad esempio, dopo aver tracciato una linea, il punto finale della linea diventa il nuovo ultimo punto indirizzato.

8. Sistemi di coordinate grafiche: il sistema di coordinate standard è chiamato sistema di coordinate *fisiche*. Nelle modalità CGA (SCREEN 1 o SCREEN 2) le coordinate *y* sono comprese tra 0 e 199, mentre le coordinate *x* spaziano da 0 a 319 in media risoluzione e da 0 a 639 in alta risoluzione. Si faccia riferimento alla discussione del comando SCREEN per le informazioni relative alle risoluzioni più elevate. L'enunciato WINDOW consente di specificare un sistema di coordinate differente, chiamato sistema di coordinate *naturali* o *logiche*. I punti possono anche essere specificati in termini di *coordinate relative*. Il comando STEP(*x,y*) fa riferimento al punto ottenuto partendo dall'ultimo punto indirizzato e spostandosi di *x* unità in direzione orizzontale e di *y* unità in direzione verticale.

Le coordinate relative possono essere usate in tutti gli enunciati che generano dei grafici.

9. Memoria: ogni locazione di memoria contiene un valore intero compreso tra 0 e 255. Questa unità di dati è chiamata byte. La memoria del computer è divisa in blocchi di locazioni di memoria chiamati *segmenti*. Ogni segmento ha una dimensione di 65536 byte. All'interno di un segmento, si può indicare una determinata locazione di memoria specificandone la *distanza* (offset) dall'inizio del segmento con un numero compreso tra 0 e 65535. Quindi, per localizzare un byte in memoria, bisogna conoscere sia il segmento che la distanza, anche se molto spesso è sufficiente conoscere solo il segmento. I segmenti si sovrappongono; ciò significa che delle stesse porzioni di memoria possono essere considerate come appartenenti a due segmenti differenti. Il segmento 0 si estende dalla locazione 0 alla locazione 65535, il segmento 1 dalla locazione 16 alla locazione 65551, il segmento 2 dalla locazione 32 alla locazione 65567, e così via. Per esempio, la trentaquattresima locazione di memoria può essere identificata come segmento 0 distanza 34, segmento 1 distanza 18, o segmento 2 distanza 2. QBasic riserva un segmento speciale, chiamato *segmento dati di default* o DGROUP, in cui memorizza le variabili, dei valori particolari come quelli relativi alla posizione del cursore, e la tabella di valori per le generazioni RND. Il *segmento corrente di memoria* viene usato in combinazione con le distanze usate nei comandi BLOAD, BSAVE, PEEK e POKE. All'inizio dell'esecuzione di un programma, il segmento corrente di memoria è il segmento dati di default, ma può essere cambiato mediante l'enunciato DEF SEG.

10. Periferiche: alcuni esempi di periferica (device) sono il monitor, la tastiera, la stampante, il modem e le unità disco. Il microprocessore del computer riceve e invia i dati alle varie periferiche ad esso collegate attraverso le *porte*. Ogni porta viene identificata da un numero compreso tra 0 e 65535.

11. File ad accesso casuale: i due metodi per la lettura e la scrittura di un file ad accesso casuale sono il metodo del buffer e il metodo che prevede l'uso di record definiti dall'utente. Quest'ultimo metodo è spiegato dettagliatamente nel Capitolo 9. Con il metodo del buffer, viene riservata una porzione di memoria (un buffer) per il file. Un enunciato FIELD specifica le variabili di campo a lunghezza fissa i cui valori sono contenuti nel buffer, i comandi LSET e RSET consentono di assegnare i valori alle variabili di campo, e gli enunciati PUT e GET spostano, rispettivamente, il contenuto del buffer in un record del file e viceversa. Le funzioni CVI, CVL, CVS e CVD vengono usate per convertire i numeri in stringhe a lunghezza fissa prima di inserirli nel buffer mediante i comandi LSET e RSET. Quando un enunciato GET ha inserito un record nel buffer, le funzioni MKI, MKL, MKS e MKD consentono di convertire le stringhe nel tipo numerico appropriato.

12. File binario: un file aperto con un enunciato nella forma OPEN *specFile* FOR BINARY AS #*n* viene elaborato come se si trattasse di una lunga e continua stringa di caratteri. Una volta aperto il file, esiste sempre una 'posizione corrente' del puntatore del file. Il comando SEEK consente di impostare la posizione corrente.

Gli enunciati PUT e GET permettono, rispettivamente, si scrivere e prelevare dei caratteri dal file partendo dalla posizione corrente. Una volta impartito un comando PUT o GET, la nuova posizione corrente diventa la posizione successiva all'ultimo carattere elaborato.

13. Etichetta: QBasic consente di identificare le destinazioni specificate dai comandi GOTO e GOSUB in due modi: mediante i numeri di riga e attraverso l'uso di etichette. Le etichette vengono create seguendo le stesse regole imposte per i nomi delle variabili e sono seguite dal segno due punti (:). Quando un'etichetta appare in un enunciato GOTO o GOSUB, il controllo del programma passa all'istruzione che si trova nella riga identificata dall'etichetta.
14. Subroutine: una subroutine è una sequenza di enunciati che inizia con un etichetta e termina con il comando RETURN. Una subroutine viene richiamata da un comando GOSUB e deve essere posizionata in un punto in cui non possa essere eseguita inavvertitamente (ad esempio, dopo il comando END).
15. Rilevamento degli eventi: alcuni eventi particolari, come la pressione di un tasto funzione o la presenza di un errore, possono essere rilevati grazie a delle apposite istruzioni. Gli eventi vengono specificati da enunciati nella forma *Evento* ON, e il controllo del programma viene indirizzato alle istruzioni appropriate per l'evento verificatosi tramite l'enunciato ON *Evento* GOSUB *etichetta.* Nel momento in cui si verifica l'evento specificato, il programma esegue il comando GOSUB *etichetta.* Per disabilitare il rilevamento degli eventi, si deve utilizzare il comando *Evento* OFF.
16. Metacomando: gli enunciati $STATIC e $DYNAMIC sono chiamati metacomandi. I metacomandi indicano all'interprete di inserire determinati codici nel programma o di considerare alcuni comandi di QBasic in un modo particolare. Dato che i metacomandi non vengono eseguiti, devono essere preceduti dalla parola riservata REM (o da un apostrofo). Ad esempio, l'enunciato REM $STATIC o '$STATIC indica all'interprete di memorizzare gli array in un modo statico.
17. Statico e dinamico: QBasic memorizza gli array in modo statico o dinamico. Le locazioni di memoria per un array statico vengono riservate al momento della compilazione e non possono essere utilizzate per nessun altro scopo. Le locazioni di memoria per un array dinamico, invece, vengono assegnate durante l'esecuzione e possono essere liberate per altre attività. Benché gli array dinamici siano molto più flessibili, richiedono un tempo di accesso maggiore. QBasic riserva la memoria dinamicamente quando un array viene dimensionato con una variabile o quando viene utilizzato il metacomando $DYNAMIC.

APPENDICE H

ALCUNE DIRETTIVE

AVVIARE E USCIRE DA QBASIC

A. Se si avvia QBasic in un computer che non dispone di un disco fisso:

1. inserire un dischetto DOS di avvio nell'unità A;
2. accendere il computer e il monitor, e attendere la visualizzazione del prompt del DOS (se viene richiesta la data e l'ora, si risponda in modo appropriato);
3. rimuovere il dischetto del DOS dall'unità A e inserire il dischetto che contiene il file QBASIC.EXE;
4. digitare QBASIC e premere Invio;
5. quando viene richiesto di inserire il disco con il sistema di aiuto, inserire nell'unità A il dischetto che contiene il file QBASIC.HLP.

B. Se si avvia QBasic da dischetto in un computer avviato da disco fisso:

1. accendere il computer e il monitor, e attendere la visualizzazione del prompt del DOS (se viene richiesta la data e l'ora, si risponda in modo appropriato);
2. inserire nell'unità A il dischetto che contiene il file QBASIC.EXE;
3. digitare QBASIC e premere Invio;

4. quando viene richiesto di inserire il disco con il sistema di aiuto, inserire nell'unità A il dischetto che contiene il file QBASIC.HLP.

C. Se si avvia QBasic dopo averlo installato su disco fisso:

1. accendere il computer e il monitor, e attendere la visualizzazione del prompt del DOS (se viene richiesta la data e l'ora, si risponda in modo appropriato);
2. digitare CD\DOS e premere Invio;
3. digitare QBASIC e premere Invio.

D. Se si dispone di un monitor monocromatico collegato a una scheda Hercules:

1. prima di avviare QBasic, eseguire MSHERC.COM dal DOS.

Nota: Se deve eseguire il comando SCREEN 3 prima di qualsiasi altro enunciato grafico.

E. Per abilitare l'uso del mouse:

1. prima di avviare QBasic, eseguire MOUSE.COM dal DOS.

F. Per uscire da QBasic:

1. premere il tasto Esc;
2. premere Alt/F/E;
3. se il programma nella finestra di visualizzazione non è stato salvato, QBasic chiede conferma prima di procedere.

Nota: In molte situazioni, il primo punto non è necessario.

GESTIONE DEI PROGRAMMI

A. Per eseguire un programma da QBasic:

1. premere Alt/E/A;

   oppure,

2. premere Maiusc-F5 (normalmente, è sufficiente premere solamente il tasto F5. Tuttavia, se il programma è stato interrotto prima del completamento, F5 fa riprendere l'esecuzione dal punto di interruzione, mentre Maiusc-F5 esegue il programma dall'inizio).

B. Per salvare il programma corrente su disco:

1. premere Alt/F/S o Alt/F/V;
2. digitare il nome da assegnare al programma e premere il tasto Invio.

Nota: Dopo aver salvato un programma per la prima volta, si possono salvare le modifiche apportate impartendo il comando Alt/F/S. Il comando Salva con nome viene usato per assegnare un nome diverso a un programma già esistente.

C. Per iniziare un nuovo programma nella finestra di visualizzazione:

1. premere Alt/F/N;
2. se il programma nella finestra di visualizzazione non è stato salvato, QBasic chiede conferma prima di procedere.

D. Per aprire un programma memorizzato su disco:

1. premere Alt/F/A;
2. digitare il nome del file e premere Invio. In alternativa, premere il tasto Tab per posizionarsi nella casella che contiene i nomi dei file, usare i tasti cursore per selezionare quello desiderato e premere Invio;
3. se il programma nella finestra di visualizzazione non è stato salvato, QBasic chiede conferma prima di procedere.

E. Per assegnare un nuovo nome a un programma:

1. premere Alt/F/V.

USO DELL'EDITOR

A. Per determinare la riga e la colonna su cui è posizionato il cursore:

1. esaminare la coppia di numeri situata nella parte inferiore dello schermo;
2. il primo numero rappresenta la riga e il secondo numero la colonna.

B. Per contrassegnare un blocco di testo:

1. spostare il cursore all'inizio o alla fine del blocco;
2. tenere premuto il tasto Maiusc e usare i tasti di direzione per evidenziare un blocco di testo;
3. per deselezionare il testo, rilasciare il tasto Maiusc e premere un tasto cursore.

C. Per cancellare una riga di programma:

1. spostare il cursore sulla riga desiderata;
2. premere Ctrl-Y;

   oppure,

1. selezionare la riga come se fosse un blocco di testo;
2. premere Maiusc-Canc.

Nota: Nella procedura sopra riportata, la riga viene inserita in Appunti e può essere incollata in un'altra posizione tramite i tasti Maiusc-Ins. Per cancellare la riga senza inserirla in Appunti, si deve selezionare il blocco e premere Canc.

D. Per spostare una riga all'interno della finestra di visualizzazione:

1. spostare il cursore sulla riga e premere Ctrl-Y;
2. spostare il cursore nella nuova posizione;
3. premere Maiusc-Ins.

E. Per usare Appunti per spostare o duplicare dei comandi:

1. posizionare il cursore sul primo carattere dell'enunciato (o del gruppo di comandi) desiderato;
2. tenere premuto il tasto Maiusc e spostare il cursore verso destra (e/o verso il basso) per evidenziare il blocco di testo;
3. premere Maiusc-Canc per cancellare il blocco e inserirlo in Appunti, oppure Ctrl-Ins per copiarlo in Appunti senza rimuoverlo dal programma;
4. spostare il cursore nella nuova posizione;
5. premere Maiusc-Ins per inserire il testo contenuto in Appunti alla posizione del cursore.

F. Per cercare del testo in un programma:

1. premere Alt/C/T;
2. digitare il testo da cercare;
3. selezionare le opzioni desiderate se sono differenti da quelle di default;
4. premere il tasto Invio;
5. per ripetere la ricerca, premere F3.

G. Per sostituire del testo:

1. premere Alt/C/C;
2. digitare il testo da cercare nel primo rettangolo;
3. premere Tab;
4. digitare il testo da usare per la sostituzione nel secondo rettangolo;
5. selezionare le opzioni desiderate se sono differenti da quelle di default;
6. premere il tasto Invio.

H. Per attivare o disattivare il comando Verifica sintassi:

1. premere Alt/O;

2. se la funzione di verifica della sintassi è attiva, il comando Verifica sintassi è preceduto da un punto. Premere V per cambiare l'impostazione o premere Esc per uscire.

I. Per verificare la sintassi di una riga (Verifica sintassi deve essere attivo):
   1. spostare il cursore sulla riga desiderata;
   2. apportare delle modifiche alla riga (ad esempio, digitare = e premere Backspace);
   3. premere il tasto freccia in basso.

Nota: La sintassi della riga viene automaticamente verificata ogni volta che si sposta il cursore da un riga appena modificata, sia premendo Invio che un tasto di direzione.

J. Per annullare le modifiche apportate a una riga:
   1. non spostare il cursore dalla riga;
   2. premere Ctrl-Q/L per riportare la riga allo stato originale.

OTTENERE AIUTO

A. Per richiedere la sintassi e una descrizione di una parola chiave di QBasic:
   1. digitare la parola desiderata nella finestra di visualizzazione;
   2. spostare il cursore sulla parola chiave;
   3. premere F1;

   oppure,

   1. premere Alt/?/I/Prima lettera della parola chiave;
   2. usare i tasti cursore per evidenziare la parola desiderata;
   3. premere il tasto Invio.

B. Per visualizzare una tabella ASCII:
   1. premere Alt/?/S;
   2. premere Tab per spostare il cursore sulla voce Codici di carattere ASCII e premere Invio;
   3. usare i tasti PgDn e PgUp per spostarsi tra il codice standard e quello esteso.

C. Per richiedere un elenco delle parole chiave:
   1. premere Alt/?/S;

2. premere Tab per spostare il cursore sulla voce Parole chiave ordinate per classe e premere Invio;

D. Per ottenere delle informazioni di riferimento:

1. premere Alt/?/S;
2. usare i tasti Tab e Maiusc-Tab per evidenziare un argomento nel riquadro Riferimento rapido;
3. premere Invio.

E. Per ottenere delle informazioni generali sul sistema di aiuto:

1. premere la combinazione di tasti Maiusc-F1.

F. Per ottenere delle informazioni sulle voci di menu:

1. vedere il punto D in *Gestione dei menu.*

G. Per ottenere un elenco delle parole riservate di QBasic:

1. premere Alt/?/I e utilizzare il tasto freccia in basso per scorrere attraverso la lista.

Nota: Per ottenere delle informazioni si una determinata parola, si sposti il cursore su quella parola e si prema Invio.

FINESTRE DI DIALOGO

A. Uso di una finestra di dialogo:

Una finestra di dialogo contiene tre tipi di elementi: rettangoli, liste di opzioni e pulsanti di comando. Una lista di opzioni è una sequenza di pulsanti di opzione nella forma () o [], mentre un pulsante di comando ha la forma <comando>.

1. Usare il tasto Tab per spostarsi da un elemento all'altro (il movimento avviene da destra verso sinistra; per spostarsi in senso contrario, si usi la combinazione di tasti Maiusc-Tab);
2. all'interno di un rettangolo, digitare le informazioni del caso o usare i tasti di direzione per effettuare una selezione;
3. in una lista di opzioni, un pulsante di opzione della forma () può essere attivato con i tasti di direzione. Un punto all'interno delle parentesi indica che l'opzione è attiva;
4. in una lista di opzioni, un pulsante di opzione della forma [] può essere attivato con la barra spaziatrice. Una X all'interno delle parentesi indica che l'opzione è attiva;

5. un pulsante di comando evidenziato viene attivato premendo il tasto Invio.

B. Per rimuovere una finestra di dialogo:

1. premere il tasto Esc;

   oppure,

1. premere il tasto Tab per evidenziare il pulsante Annulla e premere Invio.

GESTIONE DEI MENU

A. Per chiudere un menu a tendina:

1. premere il tasto Esc.

B. Per aprire un menu a tendina:

1. premere Alt;
2. premere la prima lettera del nome del menu. In alternativa, si può premere il tasto freccia in basso, o usare i tasti cursore per evidenziare la voce desiderata e premere Invio.

C. Per effettuare una selezione da un menu a tendina:

1. aprire il menu a tendina. In ogni voce di menu c'è una lettera evidenziata, o in un colore differente, che può essere utilizzata per impartire quel comando;
2. premere la lettera evidenziata. In alternativa, usare il tasto freccia in basso per spostare il cursore sulla voce desiderata e premere il tasto Invio.

D. Per ottenere delle informazioni sulle voci di un menu a tendina:

1. aprire il menu a tendina;
2. evidenziare la voce desiderata utilizzando i tasti cursore;
3. la barra di stato nella parte inferiore dello schermo mostra una descrizione della voce selezionata;
4. premere F1 per ottenere ulteriori informazioni.

E. Per esaminare il contenuto di tutti i menu nella barra principale:

1. premere Alt/F;
2. premere il tasto freccia a destra ogni volta che si vuole visualizzare un nuovo menu.

LE PROCEDURE

A. Per esaminare una procedura esistente:

1. premere Maiusc-F2 ripetutamente per passare attraverso tutte le procedure;

   oppure,

1. premere F2. La prima voce indica la porzione principale del programma, mentre le voci rimanenti indicano delle procedure;
2. usare i tasti di direzione e il tasto Invio per selezionare la procedura desiderata.

B. Per creare una procedura:

1. spostarsi su una riga vuota;
2. digitare SUB (per un sottoprogramma) o FUNCTION (per una funzione) seguito dal nome e dai parametri della procedura;
3. premere il tasto Invio. Apparirà una nuova finestra contenente il nome della procedura e il comando END;
4. digitare la procedura nella nuova finestra;

   oppure,

1. premere Alt/M/S (per un sottoprogramma) o ALT/M/F (per una funzione);
2. digitare il nome della procedura e gli eventuali parametri;
3. premere il tasto Invio. Apparirà una nuova finestra contenente il nome della procedura e il comando END;
4. digitare la procedura nella nuova finestra.

Nota: Per ritornare al programma principale premere F2/Invio.

C. Per modificare una procedura:

1. premere Maiusc-F2 fino a visualizzare la procedura desiderata;
2. apportare le modifiche desiderate;

   oppure,

1. premere F2;
2. spostare il cursore sulla procedura desiderata;
3. premere il tasto Invio.

D. Per cancellare una procedura:

1. premere F2;

2. spostare il cursore sulla procedura desiderata;
3. premere Tab/E/Invio.

E. Per inserire una procedura in un programma:
1. aprire il programma contenente la procedura desiderata e premere Maiusc-F2 fino a quando la procedura non appare sullo schermo;
2. contrassegnare la procedura come blocco. A questo scopo, spostare il cursore sul primo enunciato della procedura, tenere premuto il tasto Maiusc e spostare il cursore sull'ultimo comando della procedura;
3. premere Ctrl-Ins per copiare la procedura in Appunti;
4. aprire il programma in cui si vuole inserire la procedura;
5. spostare il cursore su una riga vuota;
6. premere Maiusc-Ins per inserire il contenuto di Appunti nel programma.

GESTIONE DELLE FINESTRE

A. Per cambiare la finestra attiva (quella che contiene il cursore e con il titolo evidenziato):
1. premere F6 fino ad attivare la finestra desiderata.

B. Per dividere lo schermo in due finestre di visualizzazione:

opzione 1: entrambe le finestre contengono lo stesso testo:
1. premere Alt/V/D.

Opzione 2: la seconda finestra contiene un'altra procedura:
1. premere Alt/V/D;
2. premere F6;
3. premere F2;
4. spostare il cursore sulla procedura desiderata;
5. premere Invio.

C. Per eliminare la divisione dello schermo:
1. usare il tasto F6 per selezionare la finestra da conservare;
2. premere Alt/V/D.

D. Per ingrandire la finestra attiva a pieno schermo:
1. premere Ctrl-F10;

2. per ritornare alla dimensione originale, premere nuovamente Ctrl-F10.

E. Per modificare la dimensione della finestra attiva:

1. premere Alt-Più per allargare la finestra di una riga;
2. premere Alt-Meno per ridurla di una riga.

MODIFICARE L'ASPETTO DELLA FINESTRA DI VISUALIZZAZIONE

A. Per rimuovere o aggiungere le barre di scorrimento alla finestra di visualizzazione (le barre di scorrimento sono necessarie solo se si utilizza un mouse):

1. premere Alt/O/S/Tab/Tab/Tab;
2. premere la barra spaziatrice per attivare e disattivare l'opzione. La presenza di una X indica a QBasic di visualizzare le barre di scorrimento;
3. premere il tasto Invio.

B. Per cambiare alcuni colori usati da QBasic:

1. premere Alt/O/S;
2. premere Tab o Maiusc-Tab per spostarsi tra le voci della finestra di dialogo;
3. usare i tasti cursore per effettuare una selezione all'interno di ogni regione della finestra di dialogo;
4. per cambiare un'opzione a due stati, come quella relative alle barre di scorrimento, premere la barra spaziatrice;
5. una volta effettuate tutte le selezioni desiderate, premere Invio.

Nota: Quando si esce da QBasic, le nuove impostazioni vengono salvate su disco in un file denominato QB.INI. Se questo file si trova nello stesso disco e nella stessa directory da cui viene normalmente avviato QBASIC.EXE, queste impostazioni saranno i nuovi default.

C. Per avviare QBasic in bianco e nero:

1. digitare QBASIC /B e premere Invio dal prompt del DOS.

D. Per utilizzare il massimo numero di righe supportate dall'adattatore video:

1. al prompt del DOS, digitare QBASIC /H e premere Invio.

USO DI UNA STAMPANTE

A. Per ottenere una copia stampata di un programma:

1. premere Alt/F/M;
2. premere il tasto Invio.

Nota: Per stampare solo il testo selezionato o il contenuto della finestra attiva, si utilizzino i tasti cursore per selezionare l'opzione desiderata.

B. Per stampare il testo presente nello schermo di output:

1. premere F4 per attivare lo schermo di output, se necessario;
2. premere Maiusc-Stamp.

Consiglio: Una volta terminata la stampa, si può rimuovere il messaggio visualizzato premendo due volte F4.

C. Per ottenere una copia stampata dell'output grafico:

1. con il DOS versione 3.0 o successiva, si esegua il programma residente GRAPHICS.COM prima di avviare QBasic, e si proceda quindi con la procedure del punto B. Gli adattatori video e le stampanti supportate da GRAPHICS.COM sono limitate. Tutte le versioni del DOS supportano la scheda CGA e le stampanti a matrice di punti standard.

Nota: In alternativa, si può svolgere questa operazione aggiungendo al programma un codice speciale che legga ogni locazione di memoria della scheda video e invii queste informazioni alla stampante attivata come se fosse un file binario. Questo metodo richiede una certa pazienza e una buona conoscenza dell'adattatore video e della stampante utilizzati.

USO DEL DEBUGGER

A. Per interrompere un programma a una determinata riga:

1. posizionare il cursore all'inizio della riga desiderata;
2. premere F9. La riga viene evidenziata e viene definita come *punto di interruzione*. Nel momento in cui l'esecuzione raggiunge questa riga, il programma si interrompe.

Nota: Per rimuovere un punto di interruzione, si ripeta la medesima procedura.

B. Per rimuovere tutti i punti di interruzione:

1. premere Alt/D/E.

C. Per eseguire un programma evidenziando i comandi mentre vengono eseguiti:

1. premere Alt/=D;
2. se l'opzione Analizza il flusso è preceduta da un contrassegno, si prema F5. In caso contrario, premere A/F5.

Nota: Per disattivare questa modalità, premere Alt/D/A.

D. Per eseguire un programma un'istruzione alla volta:

1. premere F8. Il primo comando eseguibile viene evidenziato;
2. premere F8 ogni volta che si vuole eseguire il comando successivo.

Nota: Si può visualizzare in qualsiasi momento lo schermo di output premendo il tasto F4.

E. Per eseguire un programma un'istruzione alla volta, ma eseguire le procedure senza interruzioni:

1. premere F10. Il primo comando eseguibile viene evidenziato;
2. premere F10 ogni volta che si vuole eseguire il comando successivo.

Nota: Si può visualizzare in qualsiasi momento lo schermo di output premendo il tasto F4.

F. Per continuare l'esecuzione di un programma precedentemente interrotto:

1. premere F5.

Nota: Se si apporta una modifica a un programma interrotto che ne impedisce la continuazione, QBasic consente di decidere se continuare senza considerare la modifica o se riavviare il programma dall'inizio.

G. Per eseguire un programma fino alla riga che contiene il cursore:

1. premere F7.

H. Per avviare un programma dall'inizio azzerando tutte le variabili:

1. premere Alt/R/R.

I. Per avviare il programma dalla riga che contiene il cursore senza azzerare le variabili:

1. premere Alt/D/I.

J. Per eseguire un comando dalla finestra Immediato:

   1. premere F6 per spostare il cursore nella finestra Immediato;
   2. digitare il comando da eseguire;
   3. premere il tasto Invio.

APPENDICE I

USO DI UN MOUSE

PRELIMINARI

Per usare un mouse con QBasic, è necessario eseguire il programma MOUSE.COM per caricare il relativo driver e disporre, ovviamente, di un mouse Microsoft o compatibile. Le specifiche relative al mouse e al relativo driver possono variare; si faccia riferimento alla documentazione allegata.

Se il driver è stato caricato e il mouse è stato installato correttamente, dopo aver avviato QBasic appare un piccolo rettangolo al centro dello schermo. Questo è il cursore del mouse e la sua posizione sullo schermo varia in accordo con lo spostamento del mouse sulla scrivania. Il cursore del mouse viene spesso denominato puntatore.

Se il puntatore è visibile, è possibile spostarlo su un comando di menu e premere il pulsante di sinistra per selezionarlo. Il menu a tendina verrà immediatamente aperto. Procedendo in questo modo, si può aprire qualsiasi menu e selezionare un qualunque comando. Per chiudere un menu a tendina senza selezionare nessun comando, è sufficiente spostare il puntatore del mouse in un'area vuota dello schermo e premere il pulsante di sinistra.

Se dopo aver avviato QBasic il puntatore del mouse non fosse visibile, ci potrebbero essere dei problemi con l'impostazione dei colori (il colore di sfondo, ad esempio, potrebbe essere dello stesso colore del puntatore). In questo caso, si provi a spostare il mouse in un'altra porzione dello schermo. Se si riesce a visualizzare il puntatore, si selezioni il comando Opzioni e quindi la voce di menu Schermo. Si scelga a questo punto un colore appropriato e si chiuda la finestra di dialogo.

Un altro metodo per selezionare un elemento è quello di 'trascinare' il puntatore del mouse. Trascinare il puntatore significa tenere premuto il pulsante di sinistra del mouse mentre si sposta il puntatore. Questa operazione può essere usata con i menu per evidenziare le varie voci disponibili. Si può selezionare una voce di menu trascinando il puntatore sulla voce desiderata e rilasciando quindi il pulsante.

UTILIZZARE IL MOUSE

Si faccia riferimento alla Figura I.1.

SELEZIONE DI UNA VOCE DI MENU

Si selezioni un comando dalla barra dei menu e quindi la voce desiderata. In alternativa, dopo aver aperto un menu, si può trascinare il puntatore sulla voce da selezionare e rilasciare quindi il pulsante.

CHIUDERE UN MENU

Premere il pulsante con il puntatore posizionato fuori dal menu.

SELEZIONE DI UN COMANDO DALLA BARRA DI STATO

Fare clic sul comando desiderato. Ad esempio, per eseguire un programma, posizionare il puntatore sulla voce <F5-Esegui> e premere il pulsante.

RIDIMENSIONARE UNA FINESTRA

Spostare il puntatore del mouse sulla barra del titolo e trascinare la finestra fino a raggiungere la dimensione desiderata.

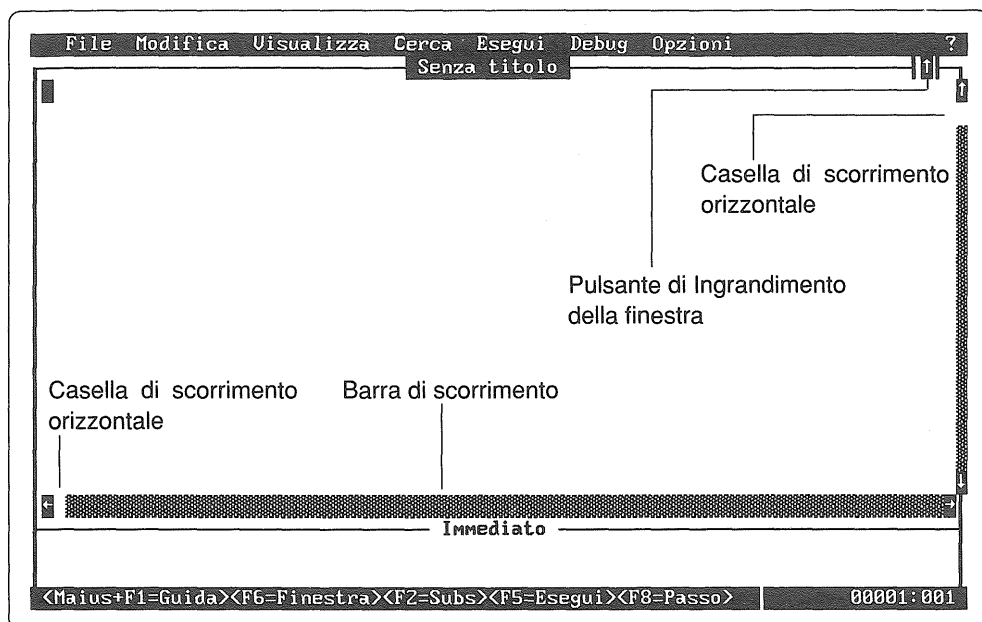



Figura I.1 *Pulsanti usati con un mouse*

APERTURA DI UN FILE ESISTENTE

Aprire il menu File e selezionare il comando Apri. Nella finestra di dialogo che appare, fare clic sul nome del file da caricare e quindi sul pulsante OK. In alternativa, si può fare un doppio clic, in rapida successione, sul nome del file.

SELEZIONE DI UN BLOCCO DI TESTO

Spostare il mouse su una delle due estremità del blocco e trascinare il puntatore fino all'altra estremità.

SCORRIMENTO VERTICALE

Spostare il mouse sulla casella di scorrimento nella barra verticale e trascinare la casella nella direzione desiderata. In alternativa, spostare il puntatore sulla freccia appropriata in una delle due estremità della barra di scorrimento nella parte destra dello schermo e tenere premuto il pulsante.

SCORRIMENTO ORIZZONTALE

Spostare il mouse sulla casella di scorrimento nella barra orizzontale e trascinare la casella nella direzione desiderata. In alternativa, spostare il puntatore sulla freccia appropriata in una delle due estremità della barra di scorrimento nella parte inferiore dello schermo e tenere premuto il pulsante.

SPOSTAMENTO DEL CURSORE DEL TESTO NELLA FINESTRA

Spostare il mouse nella posizione desiderata e premere il pulsante di sinistra.

CAMBIARE LA FINESTRA ATTIVA

Fare clic in un qualsiasi punto visibile della finestra da attivare.

INGRANDIRE LA FINESTRA ATTIVA A PIENO SCHERMO

Fare clic sull'icona che rappresenta una freccia rivolta verso l'alto, o fare un doppio clic in un punto qualsiasi della barra del titolo.

RIPORTARE LA FINESTRA ATTIVA ALLA DIMENSIONE ORIGINALE

Fare clic sull'icona che rappresenta una freccia rivolta verso l'alto, o fare un doppio clic in un punto qualsiasi della barra del titolo.

INDICE ANALITICO

D

E

H

I

K

L

M

N

O

P

THE PETER NORTON PROGRAMMING LIBRARY

PROGRAMMARE IN

David I. Schneider e Peter Norton Computing Group

Il BASIC, che è senz'altro il linguaggio di programmazione più diffuso nel mondo, è stato per anni la prima palestra per molti aspiranti programmatori: sia per la semplicità di apprendimento delle sue regole che, fatto altrettanto importante, per essere sempre stato fornito a corredo del sistema operativo MS-DOS, consentendo agli utenti di personal computer di possedere subito un tool di sviluppo per risolvere i problemi più immediati. Si è quindi trasformato nel tempo, crescendo con il sistema operativo, sono nate le prime versioni dotate di compilatore, ed oggi esistono prodotti come il BASIC Professional ed il QuickBASIC, che sono dei linguaggi di programmazione potenti e completi, invero abbastanza lontani dal BASIC standard. Con l'avvento della versione 5, Microsoft ha rinnovato il Basic fornito con il sistema operativo, proponendo il QBasic. Ora con l'MS-DOS si ha a disposizione un interprete Basic molto più evoluto, un linguaggio di programmazione strutturato, completo di tutti gli accessori per generare programmi anche complessi con una metodologia attuale. Questo libro si propone come una guida completa all'apprendimento del QBasic, sia per coloro che intendono utilizzarlo in sostituzione del BASIC standard che per coloro che sono nuovi alla programmazione in genere. La trattazione è corredata di moltissimi esempi, che descrivono soluzioni di problemi in diversi campi della programmazione: gestione di file di dati, grafica e suono, uso del mouse, programmi scientifici e matematici.

SOMMARIO

• Introduzione al QBasic • Una breve panoramica su QBasic • L'editor di QBasic • Gestione dei dati • Decisioni e ripetizioni • Funzioni, subroutine e sottoprogrammi • File di dati • Grafica e suoni • Programmi matematici e scientifici • Programmi gestionali • Funzioni incorporate • Conversione in QBasic dei programmi in BASIC standard •Il debugger di Qbasic • L'ambiente di QBasic • Parole riservate • Comandi, funzioni e metacomandi • Alcune direttive • Uso del mouse

ISBN 88-256-0444-0
9 788825 604443

L. 63.000

Cod. 1044